Extensible & Executable Lexicon Meta Model for Non-English-Like Programming Languages

Prof. Osée Muhindo MASIVI
Associate Professor
Department of Computer and Management Information System
School of Business, Rusangu University, Zambia

Abstract— Modern computer programming languages are supposed to be user friendly for every common man including non-English speakers. Unfortunately, this goal has not been reached because more than 80% of programming languages are built on English keywords and therefore, are only friendly to English speakers. Hence, programming quality as well as productivity are compromised for non-English native programmers. To address this problem, this paper proposes an extensible Meta model for programming languages (XMLPro) using local language lexicons and syntaxes. It is an executable Meta model proposing a real environment for setting up programming languages using any local languages. XML Pro allows its users to create a personalized lexicon and design a syntax aligned to their local languages grammar. On the long run, XML Pro would increase the productivity, quality, and time-to-market in programming language development. However, some improvements are needed in terms of complex grammar generation including object oriented features and windows user interface components.

Key words: Meta Model, Non-English-Like, Programming Language, Lexicon

I. INTRODUCTION

Programming language takes an important role in computer application development [1] as well as in programming learning [2] [3]. Therefore, programming languages are evolving from heavy machine level programming languages to user friendly programming languages. However, the so called user friendly programming language are built on English keywords and therefore, are only friendly to English speakers. As of now, English language knowledge is becoming a pre-requisite for computer programming [4] due to the overwhelming trend in programming languages to use the English language keywords and code libraries [5].

Consequently, when software development involves non-English developers, the team faces two language barriers: communication language between members and code language between members and computers [6] [2]. Then, programming quality as well as its productivity are compromised because programmers not only focus on program algorithms, but they also spend time to understand the English-based programming language keywords and syntaxes. Yet, computing in general and programming in particular should not be the luxury of a privileged group of people who know English. Therefore, there is a great need of non-English-like programming language for non-English speaking programmers. Common man and pupils should be enabled to program a computer using their mother tongue and regional languages as well. Since many of these regional languages are well-structured and spoken by millions of people, they can be used as programming language and as channel of communication among developers.

This paper advocates that real user friendly languages should allow programmers whose native languages is not in English to code in their local languages. It proposes an Extensible and Executable Metamodel called XMLPro for non-English-like programming language. The main objective behind this metamodel is to easily generate any local-language-like programming language lexicon. Furthermore, the metamodel permits Extensibility: new concepts may be added that enable new functionality patterns in the language or code to be expressed more succinctly, while unused elements of the language can eventually be driven out. The generated programming language focuses on very basic components of programming language such as input and output handling, operation on variables, logical conditions and iterations. The Graphic User Interface (GUI) for desktop and web application development is out of our implementation in this research.

II. RELATED WORKS

Metamodels are not a recent development, but rather a new terminology and a different focus. In this research, a metamodel is understood as a model of a programming language which describes a programming language at a higher level of abstraction and allows the same model to be used in generating different programming languages [7]. Although extensive research has been carried out on programming languages metamodels, two major characteristics of a metamodel have not been fully addressed: Firstly, previous metamodels have not entirely captured the essential features and properties of the new programming language design and code generation in terms of concrete syntax, abstract syntax and semantics. Secondly, previous metamodel architectures don't allow metamodels to be described by a single metamodel: the meta-metamodel [1] [8].

In fact, many models and metamodels have been designed for code generation for almost all programming languages [7] [9]. A number of them have translated English keywords to local languages such in Baik [6], in Tamil [10], [11] [12], [13] in Russian [14], in Bengali [15], in Korean [16] [17] in common and practical English [18], in spoken English [19], Arabic version of SQL [20] and many other languages [5]. This step though helpful, can only solve half of the problem. Since every local language can be unique in terms of lexicon and syntax, a metamodel for local languages lexicon and syntax design is still needed.

Designing solely one local programming language does not solve the problem addressed in this paper since other local language speakers will face the same challenges. Hence the need of an Extensible Metamodel. The benefit of such a metamodel is its ability to describe a number of local languages in a unified way so that they can be uniformly

managed and manipulated. This solves the problem of language diversity by allowing mappings constructions between local languages and existing languages. Furthermore, the extensible metamodel will able to define semantically rich language abstracts. Many different abstractions could be defined and combined to create new languages that are specifically tailored for a particular application domain [1].

III. METHODOLOGY

The target of this paper is a lexicon and syntax for an Extensible Metamodel suitable for Non-English Programming Languages (XMLPro). To come up with XMLPro, syntactic approach coupled to scenario based methodology [21] with textual analysis (noun-verb analysis) technique adapted from Abbott (1993), Chen(1983) and Graham (1995) have been used. An emphasis has been put on two key programming language features, abstract syntax and concrete syntax [1]. Since the task of creating a metamodel for a language is not a trivial, this paper focuses on the following basic steps enabling metamodel quality, test, simulation and experimentation among the five proposed by [1] :

- Abstract Syntax definition: describes the vocabulary of concepts provided by the language and how they relate to create models.
- Well-Formedness Rules and Meta-Operations definition: describes well-formedness rules used to validate the model correctness.
- Concrete Syntax definition: describes the concrete syntax in terms of grammar rules and validated with informal examples with some considerations of extensibility architecture.
- Mappings to other Languages construction: makes the model executable, enabling users to evaluate and execute the model.

IV. PROPOSED METAMODEL

A. *XMLPro Abstract Syntax*

In this section we describe the XMLPro abstract syntax model: the concepts in the all non-English-languages, their relationships to each other. The XMLPro is built on six major programming language concepts as suggested by Albacea (2003): character set, numbers, strings, identifiers, reserved words (operators, keywords and delimiters), expressions and statement. Three groups of elements are structured into three packages: Vocabulary (character set, numbers, strings, identifiers, and reserved words), Expressions and Statements in diagram Fig.1 [22]. This paper focuses on Vocabulary package.

1) XMLPro character set: Character set is the set of symbols called alphabet or input symbols used to build accepted words in the language [23].
2) XMLPro Words: In XMLPro the concept is "Word" which is a set of zero or more (composition) characters from the character set ("CharSet").
3) XMLPro Values: Values are atomic data values manipulated by XMLPro. They are either constant numbers or strings, or values stored in variable identifiers.
4) XMLPro numbers: Numbers are Words recognized as numbers (integer or decimal).
5) XMLPro String: String is a Word recognized as string (sequence of characters).
6) XMLPro Operators: Operators are Words recognized as operators symbols (mathematical, logic, comparison, etc.). Each operator symbol must have a key name and a correspondent program value.

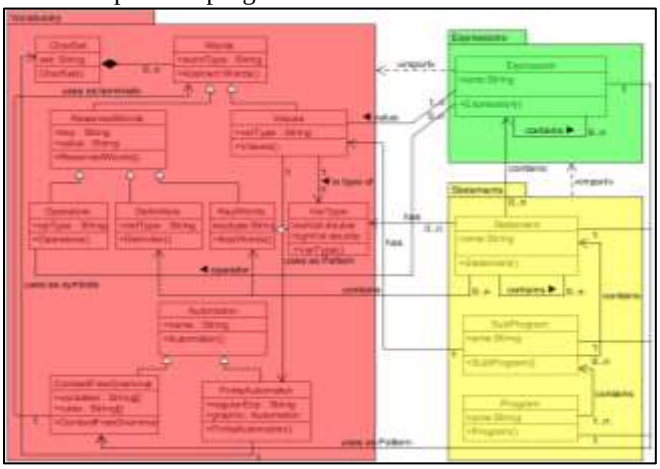

Fig. 1: XMLPro Abstract Syntax

7) XMLPro Keywords: Keywords are Words used as fixed parts of the language with a specific meaning (print, input, variable declaration, if test, iteration, etc.).
8) XMLPro Delimiters (separators): Delimiters are special Words used to mark the beginning or end of some constructs.
9) XMLPro Expressions: XMLPro expression is program phrase made of vocabulary Words to produce values.
10) XMLPro Statements: XMLPro statement is an instruction, a smallest standalone element of program submitted to the computer.
11) XMLPro Subprogram: XMLPro program is a structure containing a set of XMLPro statements for a specific task. This include procedures, functions and modules.

B. *XMLPro Well-formedness Rules*

A number of constraints apply to local programming language concepts generated by XMLPro. XMLPro itself has some meta-constraints applicable to those constraints. These constraints and rule are the rules which determine whether a model (local programming language) and the program written in that language is valid or not. These are the XMLPro well-formedness meta-rules in eXectuable Object Command Language (XOCL) [22]:

1) Character Type (CharType) is a non-empty set of character.

```
context CharType
        @constraint Non-emptyCharType
CharType->ForAll(c in
CharType|CharType.count>0)
End
```

2) A Word must have only characters defined in the character set: a character in the word must be of type CharSet.

```
context Words
```



```
        @constraint CharacterMustbeOfCharType
Word->ForAll(c in Word|c.isKindOf(type::CharType))
        End
```

3) A Reserved Word (Operator, a Separator or a Keyword) cannot be used as a Value: Sets of Reserved words (W) and Value (V) are totally disjoint (W=RUV and R∩V=ϕ).

```
    context Words
        @Constraint NotAReservedWordCanBeAValue
                ReservedWord->forAll(rw |
                Value->forAll(val |
                rw = value implies val = rw))
end
```

4) Operators (O), Delimiters (D) and Keywords (K) are disjoint sets: R=OUDUK and O∩D∩K= ϕ. They are checked in the following order: Operators, Delimiters then Keywords, say if it is an Operator, it cannot not be Delimiter not a Keyword and so on.

```
    context ReservedWords
        @Constraint OperatorsDelimitersKeywordDisjoint
                Operator->forAll(op |
                Delimiter->forAll(del |
                Keyword->forAll(kw |
                op = del implies del = op)
                del = kw implies kw = del))
end
```

C. XMLPro Concrete Syntax

This section describes the basic XBNF XMLPro grammar rules for a concrete local programming language drawn from the above abstract syntax. The result is to create an instance of the abstract syntax classes.

1) Grammar

```
@Class Expression extends Named
        @Attribure pattern:Grammar end
        @Grammar
            Expression::= Name = ExprName Paterm =
            ExprGrammar:Grammar
            ExprGrammar::=Grammar(word[]:Words,
            value[]:Value, op[]:Operation
        end
end
@Class Value
        @Attribure valType:String end
        @Constructor(type,pattern)
                self.addDaemons()
        end
        @Grammar
            Value::= valType = type paterm =
            ValPattern:FiniteAutomaton
            ValPattern::=
            FiniteAutomaton(symbols:CharSet)
        end
end
```

2) Validation

A number of different properties of the model are usefully tested by the partial snapshot in Fig. 2. These include checking the well-formedness rules and query operations by running them against the snapshot. The Fig.2 shows that an expression (:expr1) is built on a context-free grammar (:gram1), other expressions (:op2), Keywords (:key1, :key2, …) and Delimiters (:del1, :del2, …). Each grammar contains words(:word1, :word2, …), Operators (:op1, :op2, …) and Values (:val1, Val2,…). Each value is built by an automaton on character set.

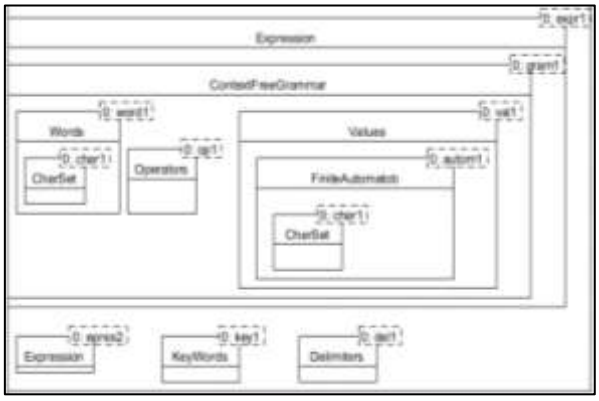

Fig. 2: XMLPro Metamodel Validation Map

D. XMLPro Mappings to C#

In this section, XMLPro lexicon aspects are made executable, enabling users of the language to simulate, evaluate by defining their simple local language. The model has been mapped experimentally to C#. XMLPro vocabulary classes have been mapped to Boundary (Dialogue), Entity and Control C# classes.

1) Dialogue Classes

Dialogue classes allow the user to enter or choose language elements to define his local language lexicon. Listing 1 shows sketch to method helping users to design local language elements patterns using Regular Languages (Finite Automata) . Designed word pattern are saved using Listing 2.

```
public string regDesign(String  reg1, string reg2,string op) {
    switch (op) {
        case "Or":
          reg1 += "|"+reg2;
          break;
                        …
        case "AtLeastOnce":
          reg1 += "(" + reg2 + ")+";
          break;
        case "DonotStartBy":
          reg1 += "^(" + reg2+")";
          break;
                        …
      }
      return reg1;
  }
```

Listing 1: Design regular expression major method.

```
  public void setElement(string file, string name, string
pattern) {
                        ….
        DataSet dict = new DataSet();
        dict.ReadXml(file);
        DataRow dr = dict.Tables[0].NewRow();
        dr["Name"] = name;
        dr["Pattern"] = pattern;
        dict.Tables[0].Rows.Add(dr);
        dict.WriteXml(file);
                        …
  }
```

Listing 2: Local Language Elements Pattern Save Method

2) Entity Classes

XMLPro Entity Classes save and retrieve the local language lexicon in and from XML files for further processing by the XMLPro control classes methods. Listing 1 shows how local language elements are retrieved from correspondent XML files.

3) Control Classes

XMLPro Control Classes implement the well-formed rules and the concrete syntax grammar and maps them to a programming language.

```
public static void initial(){
        motCles = new Dictionary<string, string>();
        operateurs = new Dictionary<string, string>();
        Separateurs = new Dictionary<string, string>();
        Automate = new Dictionary<string, string>();
                … // for keywords dictionnary
    LexiconElements lexEl = new LexiconElements();
    DataTable table=lexEl.getElemt("dictionnaire.xml");
    for (int i = 0; i < table.Rows.Count; i++)
            motCles.Add(table.Rows[i]["Mot"].ToString(),
          table.Rows[i]["Cle"].ToString());
          … // for other dictionnaries
}
```

Listing 1: Local language Elements Dictionary Feeding

Upon loading the lexicon elements pattern using the methods public static void initial()(Listing 1) the following functions are used in the lexicon analyzer method (public string lexiconAnalyser()(Listing 2) that analyses the input word, split it into lexicon element, evaluate each element, qualifies it accordingly and return it as a token in the local language and its correspondence in the target language:

- void sauterEspaces() : Evaluate character and make sure all spaces have been removed from the text to be analysed.
- bool estOperateur(out string res): Evaluate operators and return the operator as result when the evaluation succeeds.
- bool estSeparateur(out string res) : Evaluate delimiters and return the delimiter as result when the evaluation succeeds.
- bool estNombre(): Evaluate the element as number and return true when the evaluation succeeds.
- bool estTexte(): Evaluate the element as string and return true when the evaluation succeeds.
- bool estMotcleOuIdent(out string res): Evaluate the element as either a keyword or as an Identification and return the delimiter as result when the evaluation succeeds.

```
public string lexiconAnalyser(){//0. Chek sequence : operator, delimiter, Value
        …
        if (estOperateur(out tokenSuivant))
            return tokenSuivant;
        else if (estSeparateur(out tokenSuivant))
            return tokenSuivant;
        else if (estMotcleOuIdent(out tokenSuivant))
            return tokenSuivant;
                else if (estNombre())
            return "Nombre";
        else if (estTexte())
            return "Texte";
            …
    }
```

Listing 2: Token evaluation and analysis

The "pattern" used in each of the above eval() functions is set by the user through Dialogue classes and stored then retrieved by Entity classes. The following algorithm is implemented for each function:

```
Algorithm 1 : evaluateToken input : word, output : Token
pattern = dictionnaryName["pattern"];
if ( !MathcRegularExpression(word,pattern)
return false;
getToken
goNextToken
return true;
```

V. IMPLEMENTATION AND TESTS

XMLPro is implemented in MS C#.NET 2012 the MS .NET Framework 4.5. A demonstration of a sample test that exhibits how to design and use a local language programming language in "LINGALA", a local Congolese language, is presented. The language is called MonProLang and can declare, assign, input and display messages and values stored in variables. Fig 3 shows reserved word definition and Syntax design the outcome is saved in XML files.

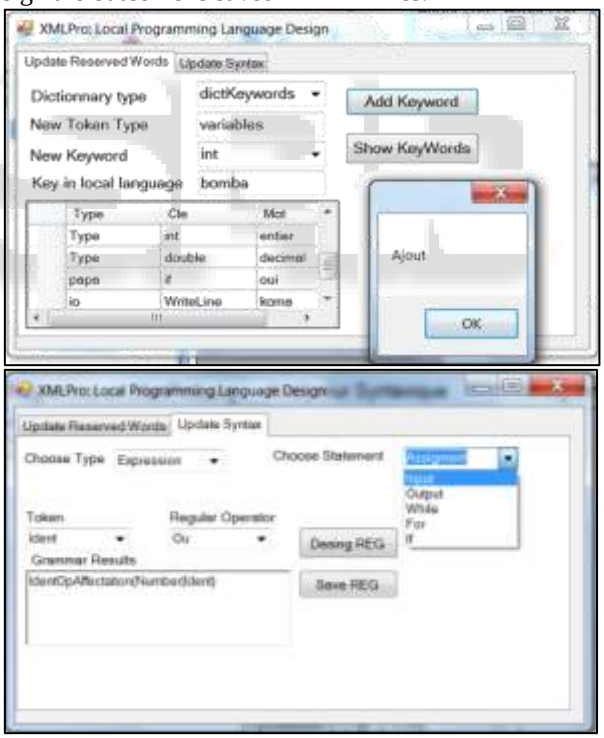

Fig. 3: Reserved word Definition (Up) and Syntax Design (Down)

Once the programming language is created, the user can then share it with is programmers who can use it. Fig 5 shows a sample program written in MonProLang, its syntax analysis and its correspondence in C# compiled by CSharpCodeProvider code generator and compiler. Syntax analysis module produces the token type, the actual keyword in the new non-English-like programming language and its location in the source code. These tokens are then confronted to the language syntax and yields the abstract syntax three

(AST). Code generator and compiler module takes the corresponding AST to generate the object code in the target language and compile it using the target programming language compiler.

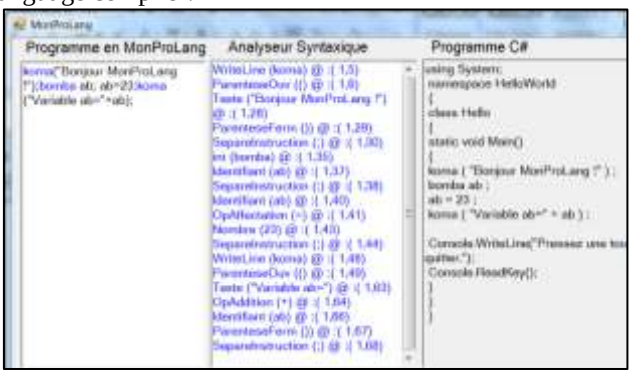

Fig. 4: Sample Local Language Programming Execution

VI. Conclusion and Discussion

Results shown above prove that XMLPro is an extensible and executable metamodel for all, including end-users, students, junior developers and teachers. Consequently, junior developers and teachers can create programming languages in a clear, familiar, systematic and rapid manner without dealing with ambiguous notations and bizarre symbols chosen inside their audience language lexicon and syntax. End users and students can then use designed local programming language without spending much time in learning, keywords rules and formal structures, and without becoming frustrated by syntax structures and programming hazards [9] [4]. On the long run, XMLPro would increase the productivity, quality, and time-to-market in programming language development since it allows mappings any local language-like programming language with C# via translation, semantic equivalence and abstraction.

As further research, other programming key technologies needing complex grammar generation are to be added to the metamodel including classes, objects and other OOP features as well as windows user interface components so that it can be adapted to support natural local languages grammar. Also a reusable set of built-in libraries including networking, file processing and database manipulation are highly needed. Finally, since the XMLPro is intended to be used by developers from speaking different languages, the user interface for XMLPro local language design is to be internationalized by supporting more human international languages such as French, German, Portuguese, Spanish, Chinese, Korean, and Arabic.

References

[1] T. Clark, A. Evans, P. Sammut and J. Willans, Applied Metamodelling A Foundation for Language Driven Development Version 0.1, Xactium, 2004.
[2] A. K. Veerasamy and A. Shillabeer, "Teaching English Based Programming Courses to English Language Learners/Non-Native Speakers of English," International Proceedings of Economics Development and Research (IPEDR), vol. 70, no. 4, pp. 17-22, 2014.
[3] A. Riker, "Natural Language in Programming An English Syntax-based Approach for Reducing the Difficulty of First Programming Language Acquisition," Brandeis University, Waltham, Massachusetts, 2010.
[4] P. J. Guo, "Non-Native English Speakers Learning Computer Programming: Barriers, Desires, and Design Opportunities," in CHI '18 Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, Montreal QC, Canada, 2018.
[5] D. Pigott, "Taxonomy and genealogy of programming languages," 2015. [Online]. Available: http://hopl.info/showstructure.prx?structureid=45. [Accessed 12 November 2010].
[6] H. Hasanudin, "Baik– programming language based on indonesian lexical parsing for Multitier web development," vol. 3, no. 2, June 2010.
[7] M. Piefel, "A common metamodel for code generation," 2006.
[8] A. Kleppe, "A Language Description is More than a Metamodel," University Twente, Netherlands, 2008.
[9] Y. A. Bassil and A. M. Barbar, "MyProLang – My Programming Language A Template-Driven Automatic Natural Programming Language," in Proceedings of the World Congress on Engineering and Computer Science 2008, SanFrancisco, 2008.
[10] N. Senthilraja, B. Amutha and M. Ponnavaikko, "Kanimozhi - a computer language in Tamil," in Tamil Internet Conference 2014, 2014.
[11] A. Muthiah, "ReseachGate," July 2009. [Online]. Available: https://www.researchgate.net/publication/45864706_Ezhil_A_Tamil_Programming_Language. [Accessed 1 November 2016].
[12] N. Senthilraja, B. Amutha and M. Ponnavaikko, "Kanimozhi - a computer language in Tamil," SRM University, Chennai, 2014.
[13] G. R. Prakash and K. Ravi, "An Overview of Swaram A Language for Programming in Tamil," Tamil Internet 2003, pp. 96-103, 2003.
[14] GOST, ""GOST 27974-88 Programming language ALGOL 68 - Язык программирования АЛГОЛ 68" (PDF) (in Russian). GOST. 1988. Retrieved November 15, 2018.," 1988. [Online].
[15] A. Kamal, M. N. Monsur, S. T. Jishan and N. Ahmed, "ChaScript: Breaking language barrier using a bengali programming system," 8th International Conference on Electrical and Computer Engineering, 29 January 2015.
[16] W. Koh, "hForth - A Small, Portable ANS Forth," Forth Dimensions, vol. XVIII, no. 2/30.
[17] Nadesiko, "Nadesiko - Japanese Programming Language (for Widndows + Turbo Delphi)," Google Code, [Online]. Available: https://code.google.com/archive/p/nadesiko/. [Accessed 11 November 2018].
[18] V. Kaplan, "A New Algorithm to Parse a Mathematical Expression and its Application to Create a Customizable Programming Language," ICSEA 2016 : The Eleventh International Conference on Software Engineering Advances, pp. 272-277, 2016.
[19] B. M. Gordon and G. F. Luger, "English for Spoken Programming," University of New Mexico, New Mexico, 2014.

[20] H. Elazhary, "Facile Programming," The International Arab Journal of Information Technology, vol. 9, no. 3, pp. 256-261, May 2012.
[21] A. Dennis, B. H. Wixom and D. Tegarden, Systems analysis & design. An Object-Oriented Approach with UML, 5th ed., California: Wiley, 2015.
[22] F. Romalho, J. Robin and R. Barros, "XOCL- an XML Language for Specifying Logical Constraints in Object Oriented Models," Journal of Universal Computer Science (J.UCS), vol. 9, no. 8, pp. 956-969, 2003.
[23] M. Sipser, Introduction to the theory of computation, 2nd ed., New York, USA: Tomson Course Technology, 2007.
[24] S. Pal, Systems Programming, New York: Oxford University Press, 2011.
[25] E. A. Albacea, Concepts in Programming Languages, 2nd ed., Quezon City: JPVA Publishing House, 2003.
[26] Abbott, "Program Design by Informal English Descriptions," Communications of the ACM, vol. 26, no. 11, p. 882–894, 1993.
[27] P. P.-S. Chen, "English Sentence Structure and Entity-Relationship Diagrams," Information Sciences: An International Journal, vol. 29, no. 2-3, p. 127–149, 1983.
[28] I. Graham, Migrating to Object Technology, Reading, MA: Addison Wesley Longman, 1995.
[29] B. Ford, "Packrat Parsing: Simple, Powerful, Lazy, Linear Time October 4-6, ,," in International Conference on Functional Programming, Pittsburgh, 2002.
[30] B. Ford, "Parsing Expression Grammars: A Recognition-Based Syntactic Foundation," in Symposium on Principles of Programming Languages, Venice, Italy, 2004.

