Assessing Easiness with Froglingo

Kevin Xu, Jingsong Zhang, Shelby Gao
Bigravity Business Software LLC
{kevin, jingsong, shelby}@froglingo.com

Abstract

Expressive power has been a well-established as a dimension of measuring the quality of a computer language. Easiness is another dimension. It is the main stream of development in programming language and database management. However, there has not been a method to precisely measure the easiness of a computer language. This article discusses easiness. Provided that a data model be easier than a programming language in representing the semantics of the given data model, this article concludes that Froglingo is the easiest in database application development and maintenance.

1. Introduction

Many database applications were written in programming language in 1960s and 1970s and they are currently still in operation. Database management system (DBMS) came to the field of database application software after 1970s. It significantly improved the productivity in the development and maintenance of database applications. However, programming language and DBMS must be employed together for a database application because of the limitation of DBMS.

Froglingo is a unified solution for information management, an alternative to the combination of programming language, DBMS, file system, and web server. It is a "database management system (DBMS)" to store and to query business data; a "file system" to store and to share files; a "programming language" to support business logic; and a "web server" to interact with users across networks. More than the combination of the existing technologies, it is a single language uniformly representing both data and application logic. Therefore, we call Froglingo system a database application management system (DAMS) [2 and 3].

Assessing the easiness of a language appears to be subjective and there hasn't been a formal method in measuring it. The introduction of Froglingo, however, sheds a light on how to view easiness more objectively. We start with the two assumptions: (1). A data model is easier than a programming language in the development and maintenance of those applications expressible by the data model; (2). If a data model is more expressive than another data model, the former is easier than the latter in the development and maintenance of database applications where a programming language is involved.

Sections 2 to 5 recall the history of the progress in the fields programming language and database management, and justify the two assumptions given above. Data model is about total recursive functions while programming language is about partial recursive functions. It is the factor considered in this article that makes the two systems different in easiness. A system having both a data model and a programming language is called a hybrid, and it becomes a unification and therefore the easiest when the semantics of the data model is equivalent to the class of total recursive function. Section 6 introduces such a unification, Froglingo.

2. Language

A language is a set of expressions used by a set of users. An expression is a sequence of words. Once an expression is presented to a user, the user reacts and produces another expression. Users use a language to communicate with each other. In other words, a user feeds an expression to another user and waits for a reply.

Talking about a language, people must be the ultimate users while computers and other devices are possible users too. A person uses a language because the expressions of the language carry his/her messages. The messages are actually the person's view of some entities in the world. Therefore, when a person is composing an expression, he/she maps a set of entities in the world to the expression; when a person receives an expression, he/she maps the expression to another set of entities. The whole set of the entities expressible by a language is called the semantics of the language.

The semantics is a way of measuring the quality of a language. We call this measurement the expressive power. The larger the semantics is, i.e., the larger size of entities a language can express, the more expressive power the language has.

The quality of a language is also measurable by using another criterion – easiness. When two languages have the same expressive power, one may prefer to use one rather than another. There is no need at this point to analyze what contribute to easiness, but we raise it because it does factor in the computer language evolution.

Two languages can be comparable in easiness only if they are equivalent in terms of expressive power. It is meaningless to compare two languages in easiness if they are two exclusive sets of semantics. But if the semantics of

a first language is a superset of a second one's, the first one can be compared with the second one in easiness when the first one is used to express the semantics of the second one.

3. Programming Language

A programming language is used by both computers and human beings. Its semantics is a subset of the entities in the world, mathematically measured by the class of computable (or partial recursive) functions, the upper-limit a computer can do.

Programming languages have been evolved from machine/assembly languages to high level languages. The unchanged part of programming languages along with the evolution is their expressive power – Turing equivalent.

What has been changed along the evolution? It has to do with the easiness of a programming language. We have been continuously working on producing an easier programming language such that we can be more productive in software development and maintenance.

A key difference of a programming language from a DBMS is its ability of representing computable functions and therefore infinite data (normally called business logic or queries) in finite expressions.

The semantics of programming language inevitably falls into the class of partial recursive functions. However we strive hard to design and to use a programming language such that all the software applications coded in the language fall into a narrowed class -- total recursive functions. The class of total recursive functions is all useful in representing software applications with the given limitation of computer.

4. Data Model

A data model is also a computer language. Its semantics is a subset of the class of total recursive functions. It is the mathematical abstraction of a database management system (DBMS) for database applications.

A data model is defined to have a data structure storing a set of data and to have a set of built-in operators under which the set is closed. By closed, we meant that an operation always terminates and returns members from the set. To emphasize the dominating role of data structure, we redefine a data model to have a data structure storing a set of objects and offering dependencies among the objects. The second definition is aimed to be equivalent to the first one while the dependency attributes the built-in operators.

By dependency, we meant that if one object in the set depends on another, then the second must be in the set as well; and reversely if the second is not in set, neither the first one. For example, an attribute in a relational table depends on its row; a child object depends on its parent object in hierarchy; and the birth to an infant depends on both mother and father. The dependency as a restriction avoids exceptions in programming language.

Dependency is also required to be decidable, i.e., data model is always able to tell if two arbitrary objects in set are dependable or not. This restriction disallows an object depending on itself (i.e., data is organized to have a cyclical loop) in the managed sets, and therefore disqualifying a computer language as a data model if there is a program in the language that doesn't terminate on a input.

With the definition above, we say that linked list and queue in programming language are the examples of data models. So are the relational data model and the hierarchical data model with the containment relationship. The traditionally called "network data model" that allow cyclical data and not clearing defining the dependencies among cyclical data is excluded from being a data model in this paper. Obviously, both Datalog and a programming language are not data models.

Practice tells us that a data model is easier than a programming language in expressing a finite set of objects (normally called business data). The dependencies that lead the managed set closed on built-in operators are the supporting factors.

5. Hybrid

Programming language defines functions by coding algorithms. Data model defines functions by enumerating properties. Although data model is preferable, programming language is inevitable.

First of all, many business data, falling into the class of total recursive functions, is desired but not able to be expressible in a traditional data model. By not expressive, we meant that some dependencies would be lost even if they were placed (decomposed) into the data structure of a data model. Hierarchical data, as a typical example, can be folded into table, but its containment relationships cannot be captured by the relational data model. As another example, dependency among the vertices in a directed graph cannot be captured in both relational data model and hierarchical data model.

Secondly, constructing arbitrary queries (business logic) on the top of managed data set needs programming language. Although built-in operators can construct a class of useful queries, they don't exhaust all the queries that are practically needed, and fall to the class of total recursive functions. For example, a query in the relational data mode cannot simply return a single attribute or a sequence of attributes out of a relational database. This has no exception even for a unification as to be discussed in the next section.

A system having both a programming language and a data model is called a hybrid.

The most popular hybrid today is the combination of the relational data model and a programming language. There are two other kinds of hybrids today: XML/XQuery and Object-Oriented approaches, partially originated from the research effort of "database programming language" in 1970s to the early 1990s. The XML/XQuery approach is based on the hierarchical data model. The Object Oriented approach is based on the network structure by adding other data structure including hierarchical.

A hybrid is easier than a stand-alone programming language in database application. A hybrid is easier because a data model is used for a part of database application. A hybrid is easier than another hybrid if the data model of the first hybrid semantically is a super set of the data model of the second one. Comparing the easiness of those hybrids based on relational and hierarchical data models is not meaningful because their semantics are overlapped and not inclusive.

6. Unification

A hybrid becomes a unification if an arbitrary total recursive function can be expressed in the data structure. In other words, the data model can be used to express all the intended business data while programming language is for business logic or queries only. It mathematically means that a unification could enumerate arbitrary software applications, as long as they are total recursive, without programming language if space was unlimited.

Given the context of easiness provided in this article, we conclude that a unification is the easiest. Froglingo is such a unification.

First of all, the EP data model [5], i.e., Froglingo without variable, is a data model. It has a data structure storing high-level functions. A high-level function takes one argument at a time and returns another function. The dependencies among the managed functions, arguments, and values are well preserved in the data structure to lead a set of built-in operators. The data structure enforces users to enter total recursive functions only [4].

Secondly, an arbitrary total function can be stored to the data structure. If the function is given by its properties, the properties are simply mapped to the data structure without losing a bit of dependencies among the properties. This is the practical way we use the EP data model. If the function is given by algorithm, its properties, a total recursive set, can be mathematically enumerated by using a Turing machine [1]. Enumerating the properties of a function in algorithm may never be practically desirable, and therefore a programming language is inevitable. It however is a mathematical claim that the semantics of the EP data model and the class of total recursive functions are equivalent; and therefore it is the easiest to manage business data as long as the properties of business data are desired to be enumerated.

Finally, Froglingo is a programming language. It has variables to express infinite data in finite algorithm. It is the practical way to express business logic.

In this article, we discussed that the data structure dominates the expressive power and therefore the easiness of a data model; and concluded that a unification is the easiest. It is seen that the traditional data models, i.e., relational data model and hierarchical data model, are special cases of a unification. It is also demonstrated that the Froglingo expression "Z <=+ A" is for the query "Is there a path between A to Z in a directed graph?".

Data structure was the terminology used at the center of the concept data model in this article. It was borrowed from the text books about programming language and commonly used to reference a method of storing finite data. It was aimed to show a intuitive view of data model without a precise mathematical definition.

Reference:

1 J. E. Hopcroft, J. D. Ullman. "Introduction to Automata theory, Languages, and Computation". Addison-Wesley Publishing Company, Inc., 1979.

2 K. H. Xu, Jingsong Zhang. "A User's Guide to Froglingo, Database Application Management System". To appear on the website: http://www.froglingo.com.

3 K. H. Xu, B. Bhargava. "A Functional Approach for Advanced Database Applications". Third International Conference on Information Integration and Web-based Applications & Services (IIWAS 2001), September 2001, Linz, Austria.

4 K. H. Xu. "EP Data Model, a Language for Higher-Order Functions". Manuscript unpublished, March 1999. http://www.froglingo.com/ep99.pdf.

5 K. H. Xu and B. Bhargava, "An Introductioin to Enterprise-Participant Data Model", Seventh International Worshop on Database and Expert Systems Applications, September, 1996, Zurich, Switzerland, page 410 – 417.