A Knowledge Representation for Expressing Simple Algorithms

Sridhar Natarajan
sridhar_n@outlook.com

Abstract

This paper is about designing a platform for creating formalized semantic representations to express algorithmic knowledge and their implementations in a high level language program. Representations are mechanisms for expressing any linguistic utterance. Improvements in human understanding of those utterances is achieved when each of those utterances are expressed using representations with the most appropriate semantic properties. This principle is applied for design of this platform. The platform can be used by computer scientists, teachers and engineers who make attempts at conveying their knowledge about a specific algorithm and its corresponding high level language program. The principal objective of the platform is aimed at improving human understanding of representations for algorithmic knowledge.

The problem at hand

Conventionally, computer scientists include logic descriptions, diagrams and source code implementations for explaining an algorithm. The descriptions express execution sequence, boundary scenarios, and logic intent using natural language constructs. The source code formalizes implementation of the algorithm. However, this approach could be very limiting for the following reasons:

1) Ambiguity present in natural language descriptions.
2) Mapping program fragments present in source code to rationale descriptions present in natural language. The reader might comprehend the overall meaning from natural language descriptions, but may fail to relate discrete components of such meaning with the constructs in the source code implementation being presented. In effect the reader might find it difficult to justify if the content he has understood is the same as the one fed to a computer as a high level language implementation

Terminology

Syntax defines the set of valid sequences of symbols in the alphabet pertaining to the language. A programming language is defined by syntax that is specified with lexical or non lexical constructs and its semantics specified with meaning. A computer program is a syntactic utterance in a formalized representation.

Semantics are constructs that represent meaning of linguistic expressions. The language can be a natural language, such as English or an artificial language, like a computer programming language. Meaning in natural languages is important to make computers better able to deal directly with human languages. Likewise meanings associated with artificial languages constructs are crucial to empower human understanding of utterances expressed in computer languages. In this paper, we will be analyzing the specific meaning representations required for rendering understandable algorithm logic and intent.

User Roles

The users for the platform can be classified under two categories. A *creator is* a person whose goal is to express algorithmic design knowledge using the platform. A *learner* is a person whose goal is to understand the algorithmic knowledge expressed by the creator.

The Expressive Platform – A Knowledge Representation Approach

We use a knowledge representation approach by constructing a *knowledge model* for the execution steps of the given algorithm. The *knowledge model* comprises formalized syntactic constructs with enhanced semantic properties. Each construct unifies syntax and semantics in a single representation that preserves the association between meaning and the actual utterance.

The Knowledge Model - Design Goals

1. *Formalized*: A knowledge model must exhaustively capture a formalized representation for the algorithm. Effectively such an utterance must be a syntactical equivalent of a high level language implementation of the algorithm.

2. *Compact*: A knowledge model must be visually compact so each interpretable segment of execution alongside its context of occurrence in the overall algorithm is exhibited within the space of a single screen.

3. *Expressive*: A knowledge model must be visually intuitive and absolve the *learner* from having to apply complex rules for understanding the utterance.

To satisfy the above goals, a knowledge model must comprise the following components:

1. **Task representation**
2. **Data representation**
3. **Code representation**

1. Task Representation

Each computationally significant execution step is a *task* of the algorithm. A task aggregates operations and the data to which such operations are applied. A **task representation** provides a view of the current task in the context of all other tasks in an algorithm. An implementation for the task representation must have the below semantic properties:

- Exhibit topological order of all tasks in an algorithm
- Exhibit sequencing or parallelism of tasks in an algorithm

1.1 An Implementation scheme for Task Representation

A Dependency structure matrix is an excellent choice for implementing the task representation. In the below sections we will decode the rationale behind choosing a DSM for implementing Task representation

1.1.1 Dependency Structure Matrix – An Introduction

A *dependency structure matrix* (DSM) is a **matrix** that shows **relationships between tasks in a project.** Formally, A DSM is a binary square matrix with n rows and columns where n is the number of tasks in a project. It has k non-zero elements, where k is the number of task dependencies in that project. Each non-zero entry is placed in the matrix against the tasks involved in that dependency.

In our context, the project at hand is the algorithm to be represented. Hence we create a DSM with tasks of the algorithm. For example, consider a project that is composed of three tasks (or sub-projects): task "A", task “B”, and task "C".

The use of graphs in managing complex structures is generally recognized. But the DSM trumps the use of graph as it provides a simple and concise way to **represent** a complex project. The DSM is amenable to analyses techniques, such as **clustering** and **partitioning** that impart powerful semantic properties to the matrix.

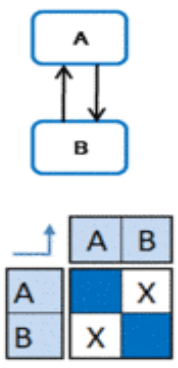


Fig: 1.1

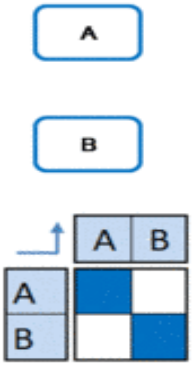


Fig: 1.2

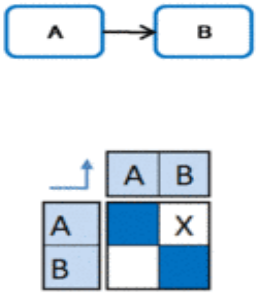
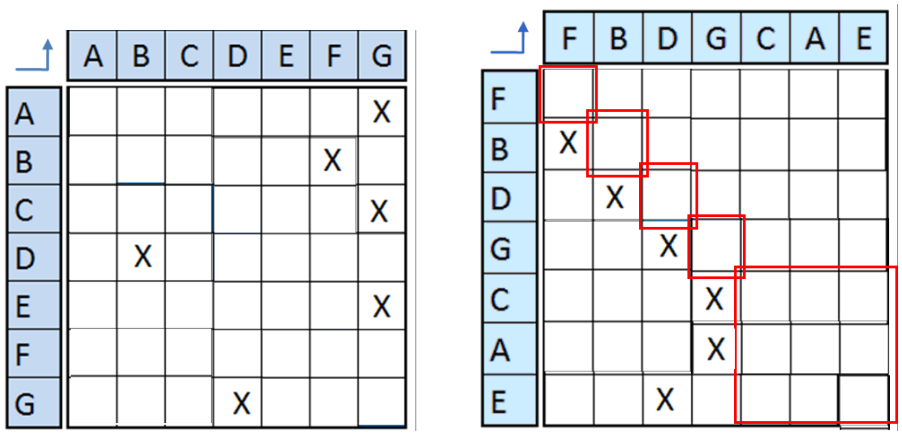


Fig: 1.3

In Fig 1.1, the DSM comprises two tasks ‘A’ and ‘B’ are cyclically connected. The strong connection between the tasks is *conveyed* by the high density of the matrix. In Fig 1.2, the two tasks are disconnected which is inferred from the empty matrix. In Fig 1.3, the two tasks have no cyclical dependency which is obvious from an empty lower triangular matrix. From reading matrix densities, we can draw inferences about the system being represented.

1.1.2 Semantic Properties of a DSM - An illustration

The semantic power of a DSM is best exploited by applying partitioning techniques over a created DSM. Let's construct a DSM for a sample algorithm with tasks and their dependencies in Fig 2.3. By applying partitioning techniques, the matrix gets transformed to the one as in Fig 2.4. In the transformed matrix, the tasks are reordered and sub-matrix boundaries are drawn using red boxes.



1.1.2.1 Reading the DSM

Without much difficulty, a *learner* can identify the following facts about the transformed DSM in (Fig 2.4).

- ✓ The upper half triangle of the matrix is empty …….. (2a)
- ✓ An empty sub matrix of size 3 is present at the bottom of the DSM against C, A & E ……(2b)
- ✓ All sub matrices except the one at the bottom are of size **1** …….. (2c)

1.1.2.2 Inferences

Going by the rules of reading modularity and sequencing, a *learner* can infer the following facts about our DSM:

Fact	Inference
2a	About all tasks, *they have no cyclical dependencies and are topologically sorted*
2b	About C,A and E, *they have no predecessors and don't depend on each other*
2c	About G, D,B,& F, *each task has to complete before the next task is taken up for execution*

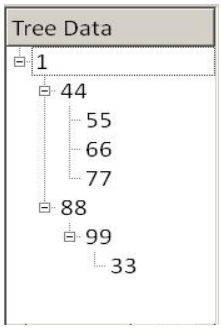
As (2a), (2b) & (2c) are visually available facts about the algorithm, the inferences listed in Table (2.5) can be observed almost effortlessly.

1.1.2.3 The DSM for Task representation

From Section 1.1.2, we know the semantic properties of the task representation to be used in our knowledge model. Now let us evaluate the semantic properties of the DSM to see if it fits as an implementation for our task representation. The partitioned DSM provides the topological order of the tasks. It also provides for distinguishing independent and sequential tasks in the algorithm. Thus we determine that the DSM can be used to implement the *task representation.*

2. Data Representations

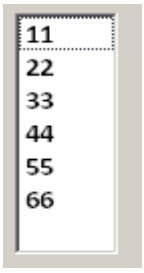
For each task, the data which is being operated is denoted by *data representations*. Data representations are visual depictions of data such as trees, lists, table, graphs used as input by functions in each task. Data representations are crucial to visualizing structural transformations to data as a result of applying an operation.



Tree Data

	1	2	3	4
1	start	data	word	algorithms
2	abc	spell	code	express
3	semantics	syntax	Hll	operation
4	human	understandable	cryptic	end

Table Data



List Data

3. Function Representation

Function representations are constructs that are needed to depict operational semantics of each task. They are best implemented by using Lisp S-Expressions.

3.1 Implementation of Function Representation

Function representations are best implemented using Lisp S expressions. The Lisp S expression is a combination of a higher order function representing the operation and the data over which the operation is applied. Abstracting data from the operations that are applied to them enables us to represent each kind of utterances (data & operations) with the most appropriate semantic representations.

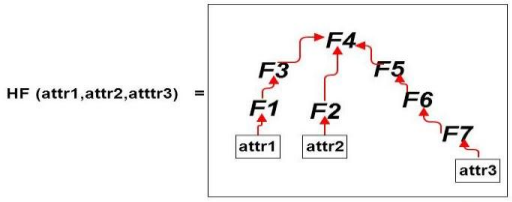
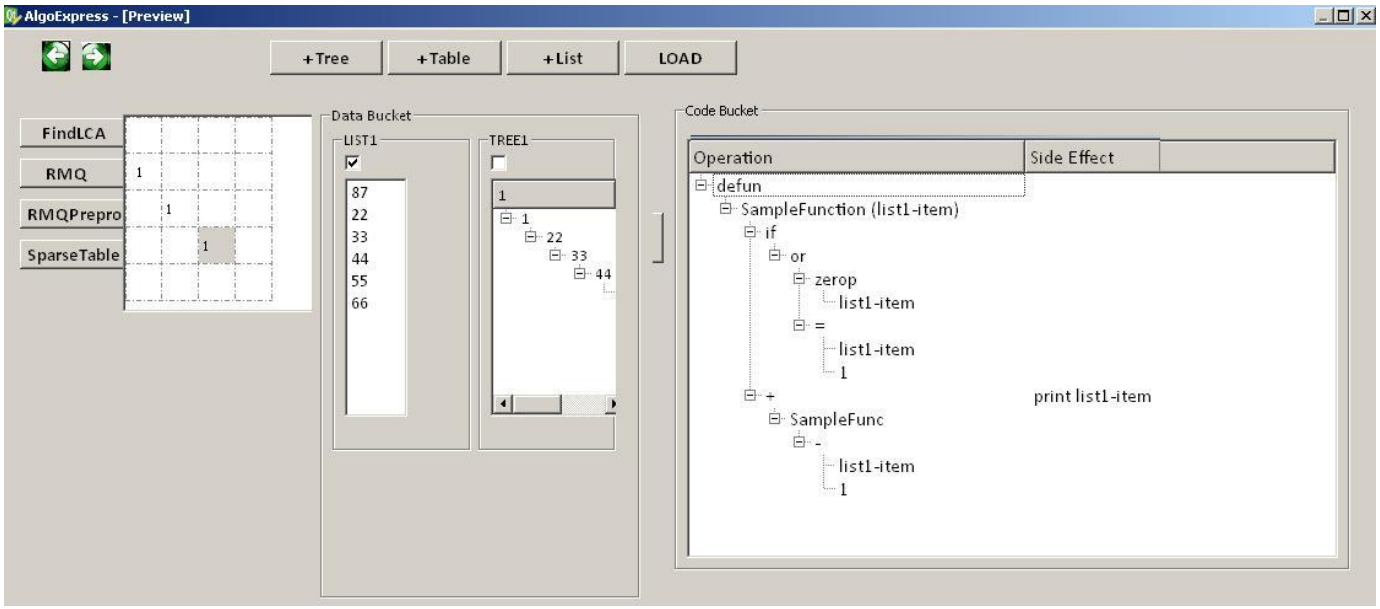


Fig 3.1

3.2 Side Effects

Lisp S expressions can represent functions that are eventually aggregated by a function for final goal which leaves no room for representing side effects**.** Incidentally, this is one of the reasons behind choosing them over imperative language statements. Usage of higher order functions to represent operations enforces explicitness in representing side effects of operations.

An Implementation of knowledge Model



A screenshot of a C++ implementation for the knowledge model is presented in Fig 4. The active task in the DSM is shaded. The data and function representations for the active task are currently displayed in the screen. The side effects of an operation are shown in the same horizontal line of that operation.

Conclusion

The paper presents lays out a scheme for implementing a system whose primary goal would be to expressively represent knowledge about an algorithm. We acknowledge the problem of having the *creator* and *learner* learn the LISP programming language to write and comprehend s-expressions. But the semantic power of lisp makes it a natural choice for representing operations with higher order functions. Benefits of human understanding might significantly differ depending on the contextual knowledge of the *learner*. The platform strives to present all relevant information necessary for understanding available with the roll of an eyeball. Improvements can be made with the level of interpreter supported and more sophisticated DSM algorithms use to showcase task relations.