Open Set of Algorithmic Characters

RENTARO YOSHIOKA, YUTAKA WATANOBE and NIKOLAY MIRENKOV
Department of Computer Science and Engineering
University of Aizu
Aizu-wakamatsu, Fukushima 965-8580
JAPAN
rentaro@u-aizu.ac.jp, yutaka@u-aizu.ac.jp, nikmir@u-aizu.ac.jp

Abstract: Programming in pictures (or filmification of methods) is an approach where pictures and moving pictures are used as (super-) characters to represent algorithms. There are two types of pictures. Compound pictures define algorithmic steps and generic pictures define the contents of compound pictures. In this paper we focus on generic pictures by providing their overview, classification and examples. We also show how the syntax-semantic gap can be bridged by the pictures and how the readability and understandability of picture-based programs are realized. In addition, the scalability of computation represented by the super-characters and the expandability of these character sets are considered.

Key-Words: CyberFilm, Filmification, Algorithmic Characters, Programming Language, Algorithms, Pictures

1 Introduction

For ancient people, drawing pictures to represent objects and actions was a “natural technique.” Pictures were easy to understand. However, available technology at the time was insufficient to support this technique. Step by step, picture languages were replaced by languages where signs represented not objects and actions but syllables, phonemes, and sometimes words. Ancient hieroglyphs were transformed into new hieroglyphs and characters for demonic handwriting. As a result, current writing systems are in principle the representation of spoken languages rather than a direct representation of data/knowledge. We can only imagine what characters would have been introduced if people, 20-30 thousand years ago, possessed computers of the current level. In any case, one of the main imperfections of the sign system evolution is the great gap between sign expression syntax and corresponding semantics. So, it is evident that a new system of signs and abstraction that can bridge the gap between syntax and semantics is necessary, and that such system of signs should take the features of the real world, human beings and modern technologies into account. LabVIEW programming environment from National Instruments [1] and Algorithmic Alphabet from Sentrana [2] are good illustrations. For promoting such necessity we are also developing a new system of characters (we call them super-characters) and special constructs to represent algorithms.

An algorithm is considered in the traditional manner: as a precise plan of actions specifying how to solve some problem. Usually such a plan is based on a finite set of instructions and constructs defining a partial order of execution of instructions. Though our focus is primarily on computational algorithms that begin with input values and yield output values in a limited number of steps, we consider the plan of actions as a set of answers to the where, when, what, why and how of the action to be done. This means that in addition to conventional constructs of programming languages, we must consider new forms of representation/specification to realize algorithm readability and understandability.

To answer the above mentioned questions, we consider algorithms as activity in 4-dimensional space-time. To define an algorithm, a space of activity, which is some structure (specified by their shapes and sizes) and their possible combinations, as well as attributes (parameters) representing the space (subspace) states is defined. Then activities of traversal schemes within the space structures are defined and concrete computational operations related to the traversal schemes are defined. Finally, we specify input-output operations representing (displaying) initial, final and possibly intermediate states of the space structures (substructures). All these specifications are arranged into four sets of compound pictures (algorithmic cyberFrames) and the sets are arranged into an algorithmic cyberFilm.
Each set of compound pictures is a special view of algorithmic features. A view is represented by its own set of generic pictures to be as self-explanatory as possible. In addition, these generic characters are supported by graphical annotations to enhance clarity of their meaning (embedded clarity).

Programming in pictures (or filmification of methods) has been applied to a great variety of algorithms including sequential and parallel matrix multiplications, solution of algebraic and partial differential equations [3-4], cellular automation-like algorithms [5], as well as algorithms on trees, pyramids, particles-in-cells, etc. [6-10]. A large set of algorithms on graphs and a corresponding library are presented in [11-12]. Some aspects of the visualization of input/output operations are considered in [13].

In this paper we focus on generic pictures by providing their overview, classification and examples. We also show how the syntax-semantic gap can be bridged by the pictures and how the readability and understandability of picture-based programs are realized. In addition, the scalability of computation represented by the super-characters and the expandability of these character sets are considered.

2 Overview of the cyberFilm concept
A cyberFilm is a set of series of compound pictures (frames). A picture is to represent a view (a feature) of an object or process. A series of pictures is to represent a multiple view (a number of features) of an object or process. A multiple view is to make the corresponding object or process be self-explained. A number of features presented by multimedia frames are assembled into a structure (cyberFilm). Usually four types of multiple views are considered: 1) algorithmic skeleton view to show data structures and dynamics of computational schemes, 2) variables and activities view to present variables attached to the space structures and formulas used in the schemes, 3) input/output view to visualize the interface operations, and 4) integrated view to show algorithmic features from the above-mentioned views in a compact view. An appropriate series of corresponding frames is extracted and presented through the right editor/browser panels at the right time. An independent visual language is used to represent the different series features of frames. As a result, the programming environment for making cyberFilms is a cluster of four mutually supplemented languages to define four different views of each software component. Each language uses its own set of self-explanatory pictures and super-characters. In these languages, open sets of visual constructs and super-characters are used to define the contents of the cyberFilm frames and their translation into executable codes. In addition to the constructs and super-characters, special background pictures and symbols, as well as auxiliary pictures are used to play a role of embedded clarity annotations for variables, operations, cyberFrames, cyberScenes and cyberFilm as a whole.

3 Classification of algorithmic super-characters
Majority of natural languages are based on very limited numbers of characters, for example, English is based on 26 and German – on 30. A special case is Chinese language including up to about 60,000 characters (though in everyday life only 3-4 thousand are used). Chinese people cannot write all these characters, but can guess their meaning if they are written. A lot of efforts should be done to memorize the characters and recognize meaning of corresponding words and expressions. Majority of conventional programming languages include English characters and some set of special symbols and constructs. Though the total number of the characters and symbols is not as huge as in Chinese language, the recognition of the program meaning and its correctness is extremely difficult. Mental simulation and abstractness involved are main points in bridging the syntax-semantic gap.

In our approach, each algorithm is represented by four different view languages supporting a concept of self-explanatory components. In addition, a “text” of each view is based on its own set of super-characters which are supported by embedded clarity annotations. The super-characters are to bridge the semantic-syntax gap on very local or regional levels; so, if the existing super-characters are not expressive enough for the meaning understanding, a new super-character with its embedded clarity annotations can be introduced. In other words, we consider the set of super-characters as an open library of items which do not request rote memorization from the users. The first direction of the item classification is their relation to four languages above-mentioned. The second one is their relation to answering the where, when, what, etc. questions.

Where category is three sets of the super-characters: a set of images representing space structures, “natural (on-default)” substructures and mask substructures, a set of images representing nodes
(positions) in the structures (they can be related to moving particles inside the structures), and a set of icons representing “index expressions” of variables involved in the specification of operations and images showing control lines (curves) positions.

When category is also three sets of the super-characters: a set of animations representing traversal schemes (fronts of computation) on the space structures and specific moments of these dynamical processes (they can include specific situations with particles), a set of images representing patterns of the space node states and animations of the state change, and a set of icons specifying waiting events (signals) in nodes.

What category is four sets of the super-characters: a set of images (including compound icons) representing computational constructs of nested, iterative, etc. types, a set of icons representing terminal and non-terminal operations and procedures, a set of images specifying inputs/outputs and features of corresponding data, and a set of icons clarifying essence of variables involved. In fact, this category, as well as others, has one more set of items to be used for embedded clarity annotations. These items are represented, if necessary, by various combinations of pictures, animation, sound, voice or text to explain meaning of corresponding constructs and data.

Why category is three sets of the super-characters to define conditions for actions related to space (including open positions and masked domains), time (including internal and external events), and features of data (including values of variables).

How category is four sets of the super-characters to define sequential, parallel, centralized and hierarchical forms of activity. They are sets with items for sequential and parallel traversal schemes, operations on so called external observer’s node, and for constructs with non-terminal operations.

In the next sections we provide examples of where, when and what category super-characters and some specific details related to their roles in the algorithm specifications.

4 Where category super-characters
The first set of this category includes images representing space structures, their “natural” substructures and mask substructures. Typical examples of the space structures are 1D, 2D and 3D meshes, trees, pyramids, and graphs. Some of these structures can be combined with particles. Fig. 1 depicts images to specify pyramid space structures.

These images allow the introduction of practically any types of pyramids including full-sized, top-less, static and dynamic shapes, as well as pyramids where a parent node can have 2, 3, 4, 8 and 16 children nodes, or pyramids put on a 3D-mesh structure. On default, the pyramids are considered as scalable structures where the number of pyramid layers and the size of its top-less part are changeable parameters.

The “natural” substructures of a pyramid are the bottom layer, the top layer, the peak node and a set of the layer perimeters. These substructures are convenient for the declaration of variables representing some specific attributes of models under consideration or just for storing some intermediate results.

Mask substructures are items to introduce necessary irregularity in the space structures and, in spite of that, to keep control of complexities appeared. They are to block (skip) computations in the masked domains. Fig. 2 shows examples of mask substructures related to pyramids.
The top-left image is to fully mask computation on a layer, the top-middle is to partially mask computation on a layer, and the top-right is to mask computation on a sub-pyramid. Other images are to show specific masking on layers. In general, masking super-characters are good for bridging the gap between an application model and its specification by the introduction of “islands” of real world irregularities, especially, on various boundaries. In many cases, they are transparent alternatives to conventional “if-then-else” operations.

The second set of where category includes images representing nodes (positions) in the structures (they can be related to moving particles inside the structures). Fig. 3. depicts examples of such images. Item 1 shows pairs of nodes involved in computation on a multi-stage network, item 2 depicts a 2D structure of nodes in a 3D mesh, item 2 shows an 1D structure of nodes and a boundary set of nodes in 2D structure. Special cases are for item 4 and 5. The first of them is to show a node in matrix C (2D structure) where elements of matrix A and B meet each other within a systolic process of their movement. The second one is to show nodes of activity within some computation on particles (with representation of particle positions in 2D space and positions on a corresponding tree structure).

In fact, a number of node domains (with different colors or filling nodes) can be simultaneously highlighted. The same colors or fillings are used to represent the same activity. So, in expressions (formulas) defining such activities, icons of corresponding node circles are used instead of conventional index expressions to point the positions of activity. These icons are from the third set of where category that also includes items for representing some neighborhood of a node highlighted.

In addition to such references, there are so called control lines (or curves) which positions can also be used for pointing the positions of activity. Fig. 4 presents six images (cyberFrames) where horizontal and vertical control lines (dashed lines) are moved within a matrix-vector space structure. These lines are used to simplify understanding of computational processes and domains of activity. Icons representing them can also be used as pointers of corresponding positions.

5 When category super-characters
Super-characters of this category are to specify dynamical features of algorithms and corresponding activity in time. The first set of the items is related to animations representing traversal schemes (fronts of computation) on the space structures. Fig. 5 shows examples of images representing traversal schemes on pyramids.
The top-left image presents a process of layer-by-layer computation starting at the bottom layer and finishing at the peak node. The top-right image presents a similar process, but two neighboring layers are involved at each step. Our intuition hints that bottom-left image is to present a process of layer-by-layer computation combining both top-bottom and bottom-top schemes.

Fig. 6 shows other examples of images representing traversal schemes on 2D structures.

For example, the bottom-left image specifies computation of column-by-column type with a combination of left-right and right-left schemes. In spite of our intuition, for these images, as well as for the above mention ones related to computation on pyramids, some doubts about precise meaning can be appeared. To clarify the situation, special animations are prepared for each traversal scheme. Fig. 7 depicts a series of frames representing the animation for the bottom-left images of Fig. 6. From this frame we can see that the u-turning process uses two frames related to computation on the most right column.

For some types of traversal schemes on complex space structures (for example, on general graphs) their understanding is supported by a few animations to avoid imperfection of intuitive views.

The second set of the when category items is related to images representing patterns of the space node states and animations of the state change. Fig. 8 shows an example of a state of 2D array of cells (in the left part) and a set of substitution rules to specify the state evolution (in the right part). The substitution rules (of two patterns each) are applied from left-to-right and performed on all cells where they are applicable. It means that if a top pattern is discovered in the array, it is substituted by corresponding bottom pattern. To prioritize the rules and avoid non-deterministic results, some additional icons can be involved.

Fig. 8 depicts a series of frames representing the animation for the bottom-left images of Fig. 6. From this frame we can see that the u-turning process uses two frames related to computation on the most right column.

The third set of the when category items is related to icons specifying waiting events (signals) in nodes and icons defining the space structure transformations. An example of such icons is later presented and explained in Fig. 11 where various types of the node flashing (highlighting) are presented.

6 What category super-characters
Super-characters of this category are to specify compound constructs of nested, iterative, etc. types,
and to define terminal and non-terminal operations and procedures, including inputs/outputs and features of corresponding data, as well as to clarify essence of variables involved.

The first set examples of these items are presented by Fig. 9. There are three compound constructs representing nesting formats of a few basic constructs. In the first (from top) compound construct three traversal schemes on a pyramid, 2D and 3D structures, respectively, are integrated. A format of the integration is presented as a “caption.” In addition, some activities (operations) of terminal (in a form of circles) and non-terminal (in a form of squares) are requested within the constructs. Terminal operations are represented by “formulas” to be directly translated into executable code. Non-terminal operations should be first disclosed through other constructs. The second and the third compound constructs at Fig. 9 disclose non-terminal activities pointed in the first compound construct. But themselves, they request specifications of one non-terminal and five terminal operations.

The second set of the super-characters of the what category is icons for the terminal operations. Fig. 10 depicts examples of such type operations.

In these examples we can see fully highlighted circles of different colors (supported by different numbers to distinguish colors). These circles are used as links to activities in traversal schemes and as “index expressions” for variables involved. In other words, they show positions of corresponding activity in the space structures and where variables can be changed. In addition to fully highlighted circles we can see contour highlighted ones. They points positions where corresponding variables can be read. The variety of highlighting forms applied for nodes of the space structures is presented by Fig. 11. Other icons, important for mentioning, are the following: rhombuses to define the formula templates, a table after the summation sign and pattern icons pointing the neighborhood of variable positions involved in summation, as well as icons presenting types and order of space nodes where data are taken from. P in the second formula means a polynomial and the precise form of it is displayed on user’s demand. An icon of the counter highlighted node with a vertical stroke in the third formula means a reference to the node number (horizontal coordinate) in the space structure. This set of super-characters also includes a variety of icons representing high-level operations or procedures. Fig. 12 shows examples of three formulas with such icons for “creating heap,” “take data from heap” and “growing sorting” operations.

The third set of the super-characters of the what category is images for the input/output operations. Fig. 13 depicts examples of such type operations.

The top-left item is to specify 4-level interface of a
multiple choice type, the next to the right is to define displaying data on a monitor, etc. Precise meaning of each item is provided on demand.

The what category set of super-characters includes items to specify a variety of procedures for data visualization and sonification. It also includes the super-characters to define some features of data which should be satisfied before the use in the algorithm and features of data which should be satisfied before output operations.

The fourth set of the super-characters of the what category is formed by images to provide additional information about variables declared as application model attributes. They are to show application meaning of variables (velocity, mass, distance, energy, etc) and corresponding units of measurement (km/h, kg, km, electronvolt (eV), etc.).

![Fig. 11. Types of the space structure node highlighting and their functionality](image)

<table>
<thead>
<tr>
<th>symbol</th>
<th>animation</th>
<th>name</th>
<th>write</th>
<th>read</th>
<th>control</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td></td>
<td>change</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(write)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(b)</td>
<td></td>
<td>take</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(read)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(c)</td>
<td></td>
<td>select</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(d)</td>
<td></td>
<td>wait</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(e)</td>
<td></td>
<td>continue</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(f)</td>
<td></td>
<td>transform</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(g)</td>
<td></td>
<td>transform-select</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Fig. 12. Examples of super-characters for high-level operations](image)

![Fig. 13. Examples of super-characters for specifying input/output operations](image)

7 Conclusion
An approach for developing a set of high-level characters oriented to algorithm specifications has been presented. These characters are used within the cyberFilm programming environment where programming in compound pictures (cyberFrames) is used for defining algorithmic steps. In this paper the focus has not been on grammar (that is on how to compose a cyberFrame), but on the algorithmic “alphabet” itself. An overview of the super-characters and their classification has been provided. Specific features of a number of them have been considered and further development of the super-character set has been indicated. An important aspect of this development is to create an annotation method to
enhance the clarity of each super-character. We are sure that programming in pictures will be useful not only for readability and understandability of the computational algorithms specified, but also for the introduction of new aspects of algorithmic aesthetics and for the involvement of new groups of users who can apply algorithm specifications to represent their activities.

References: