SEMANTIC NETWORK IN SIGNAL PROCESSING AND IDENTIFICATION THEORY

A.E.Barabanov
Saint Petersburg State University
Saint Petersburg, Russia
e-mail: andrey.barabanov@pobox.spbu.ru

Abstract

An automatic partitioning algorithm is described that decomposed an arbitrary program text into a tree structure of objects that are extracted by analysis of information flows. The algoritrhm was implemented for standard functions in signal processing. The aim of the decomposition is development of the semantic network that is a self-tuning solver in a specific scientific field.

1 Introduction

Problems of signal processing and system identification theory appear everywhere in the modern communication systems, in digital control of dynamic plants and in corresponding applications. The basic notions of the theory has become a standard set of definitions and main algorithms. The algorithms are currently implemented in the standard toolboxes that have a library structure.

A semantic network is a solver of problems in a specific scientific field that is an open system without a head node, contains a friendly interface with a fixed language and includes a self-tuning adaptive subsystem that optimizes its performance. A solver does not make an exhaustive search among admissible ways of implementation of basic functions. Instead, an active node of the network makes a decision about the next step itself by the results of computations in this node. This decision is influenced by statistics of successful solutions of other problems in the same scientific field.

Since the network changes its structure adaptively, the syntax of the knowledge representation in memory must be friendly to its semantics. In particular, a script of any standard mathematical algorithm must be written in a single node that can be called from other algorithms. Thus, any algorithm is represented in the network as a sequence of calls to basic nodes with formal parameters that are calculated previously.

A transformation from the library structure to the network structure is effort consuming and it hardly can be done manually. A script from the library are to be decomposed into standard parts that are called objects. An object must have its own mathematical sense that is not connected to the program implementation.

The automatical program decomposition into objects is made by the IPBrowser. In this paper, a new implementation of the IPBrowser is presented. The main task of the IPBrowser work consists in partitioning the program text into parts (IP-blocks or IP-objects) that have their own mathematical sense, which does not depend on

architecture or hardware implementation. The object selection criteria and syntax tools of automatic analysis of the program are given in Section 7.

The IPBrowser has to partition the program into several blocks and it also has to identify those blocks with objects in a library, if exists. It is assumed that the standard signal processing functions are stored in the library. Each function in the library is supplied with mathematically equivalent algorithms of its implementation and with other additional information, which is useful for hardware optimization.

2 Underlying Ideas of Object Extraction Algorithm

There are several problem statements of object selection depending on libraries, experts, semantic comments and a research field. A brief description of object features, problem statements, general rules are given in Section 7.

Assume the worst case initial condition where no information about program semantics is available. The object extraction algorithm is based on the information flow analysis and consists of two stages.

1. The intensive lexical and grammar analysis of the program text that gives information about program structure and variable flows.

2. On the second stage the data prepared at the first stage are processed and a set of candidates to objects is determined. It is recommended to look through this set and to eliminate superfluous candidates, if necessary. This stage consists of 3 steps.

   (a) A list of all admissible segments is formed by elimination of those segment which destroy the C-program structure stored in the file of explicitly declared C-blocks.

   (b) For each admissible segment the numbers of inner variables, input variables, output variables are computed. These sets of variables can be considered as the corresponding variables after this segment is replaced by a function call. For instance, an inner variable of the segment is an inner variable of the function.

   (c) An iterative procedure sequentially extracts segments with the maximal number of inner variables under some weak conditions. This program contains some thresholds. The most important threshold is the minimal number of inner variables N_{inner}. After a segment is found the program tries to split it again, so the relatively small segments that cannot be split become final objects.

3 Syntax Analyses and Parsing Tree

Various grammar constructions can be recognized by the automatic analysis of the source text. It was found that a few parameters like the number of variable occurrences

and indicators admissible blocks are sufficient for a perfect object selection if a source program is well written and can be mathematically interpreted clearly. For the general case more general tools are useful.

3.1 Required Tools.

The input data of the object selection algorithm at the stage 2 consist of two files with integers that are processed by the optimization algorithm. These files can be formed manually, although this is not a good solution. Moreover, very important lexical features of the source program should be also taken into account. The following tools are necessary for effective automatic object identification:

- a lexical analyzer and grammar rules which detect C-statements and fill a list of variable occurrences;

- a program which establishes identity of two blocks if the difference in identifiers names only;

- an algorithm of equivalent transformation of the program for a search of a given lexical template. For instance, this can be an admissible transposition of assignment statements for the sake of a correct recognition of a given string of statements, or an admissible change of a variable that breaks a syntax connection with its previous occurrence.

3.2 The Tree Structure.

The structure of a source program is recognized by a parser. After the parsing is completed each variable, expression and each admissible segment of the program is represented by a token of a specific type. Tokens are aggregated into a graph structure, which used to be a tree.

The parsing tree consists of tokens that have a type only. Other information is attached in their semantic values. The tree makes it much easy

- to detect an identity of two different blocks that have the same semantic content but different notation;

- to make a list of all admissible segments in the program. All these segments are represented by tokens in the tree structure, which correspond to statements of special kinds.

3.3 Additional analysis of information flows.

Assume two different cycle operators begin with the same string:

"for (i=0; i¡n; i++) ..."

Obviously, the variable i is semantically different in these two strings although the lexical analyzer reports the identity of these occurrences. This identity is indifferent for a compiler. But for the analysis of data flows it is important whether the next variable i is connected with the previous i.

The construct $i = 0$ in the cycle operator header is recognized by the parser. Since the right-hand side contains a constant one can conclude that the variable i is new and it has nothing in common with previous occurrences of i. In the more general case the right-hand side of the assignment operator is not a constant but an expression. Then the parsing of this expression accompanied by a recurrent analysis of variables included in the expression will show whether this variable is new or not. In particular, consider the well-known program of the variable values transposition: $tmp = x$; $x = y$;

$y = tmp$. Here the variable y is not new because it replaces only the input variable x that was lost after the operator $x = y$.

Another opportunity of useful analysis of the parsing tree structure is a verification of admissibility of a program transformation and then making the transformation with the aim to superpose two segments.

4 Software for Parse

The well-known tool "gcc" compiles an arbitrary C-program. This tool contains the very important function "yyparse" which is built preliminarily by the program "Bison". The input file of "Bison" contains lists of tokens and token types, a list of grammar rules with their semantic values attached in the form of C-statements, and some auxiliary C-functions and definitions.

Bison inserts semantic values attached to a grammar rule to the output function "yyparse". They execute after reducing the token stack by the rule during parsing the source C-program. Simultaneously a new "nonterminal" token is defined in the parsing tree, which represents the grammar rule implemented.

This standard construction is currently used for extraction of identifier's occurrences, for recognition of the beginning and the end of each C-block, and it can be used for other useful analysis of the parse tree.

5 Currently Implemented Additional Parse Analysis

The input grammar file with description of C language and the lexical analyzer both included in "gcc" have been augmented by the following commands that do not destroy the C-compiler built by "gcc". The lexical analyzer was augmented by the commands, which determine a list of all occurrences of variable identifiers in the source program. The process of list computation is controlled with special flags that are added to the semantic values of the grammar rules in the input grammar file. For instance, special commands recognize whether a variable identifier in the cycle operator must be considered as new or it is connected with the previous occurrences.

As a result, the parsing function "yyparse" in the output file of Bison can compile an arbitrary C-program and as well it records to the file a list of all variables together with the lines of their occurrences. Semantic values of the grammar rules were augmented by operators, which record to the file numbers of the first and of the last lines of the C-statements, which are explicitly written in the program. These statements include conditional operators, cycle operators and each block defined by { and }.

The two new files defined during parsing are inputs for the multi-criterion optimization recurrent function, which extracts new objects.

6 Example. Object Extraction in FFT - Radix4

The object selection algorithm was tested on a number of standard functions of the signal processing theory. They include the half-rate spectral vocoder and the FFT implementation by means of the Radix-4 algorithm with the additional carriers rotation.

The C-code in the file "Radix4.c " was processed by the modified version of "gcc" program with the additional operators described above. As a result, in addition to the standard object and executable files the new files with variable occurrence and with C-blocks bounds were defined. The last two files were processed by the object extraction program. The output of this program contains the line numbers of the beginning and of the end of each object extracted.

The number of objects extracted is dependent on several thresholds in the recurrent multi-criterion optimization program. The most important is the threshold N_{inner} for the minimal number of inner variables. A program segment as a candidate to an object is ignored if the number of variables that are used in this segment but not outside it is less than the threshold N_{inner}.

The result for the source file Radix4.c with the threshold $N_{\text{inner}} = 2$ contains totally 20 extracted objects. The user can accept or remove some of them. It appeared that 16 extracted objects describe the standard mathematical functions. The list of these functions is given below.

1. Cross summations, the first part of Butterfly operation: Object 2.
2. Standard Butterfly operation for 4 samples: Object 5.
3. Butterfly operation for 4 samples with rotation and scaling: Object 6.
4. Vector 4-point Butterfly operation with rotation: Object 7.
5. Dynamic scaling factor update between vector Butterfly operations: Object 8.
6. The Butterfly operation for 2 samples (without multiplications, that is used in the last FFT iteration): Object 9.
7. Shift array: Object 10.
8. Loop on blocks (the vector 4-point Butterfly operations): Object 11.
9. Vector 2-point Butterfly operation: Object 12.
10. Complex conjugation (for input data and for output data; this switches between direct and inverse FFT): Objects 13 and 17.
11. Final integer scaling after FFT: Object 14.
12. Initialization of the scaling gain: Object 15.
13. One etage of the vector 4-point Butterfly operation: Object 16.
14. The full FFT with 4-point Butterfly and with scaling: Object 18.
15. The final bit reverse transposition in FFT: Object 19.

If the threshold N_{inner} increases than the number of extracted objects diminishes. For N_{inner}=3 only 12 objects are extracted. Most of them coincide with that obtained with the threshold N_{inner}=2. Totally 8 extracted objects have mathematical sense. They do not focus on details, some of them correspond to the above objects with the numbers 2, 6, 7, 8, 12, 14, 16. The thresholds variations can be used for control the analysis of the "narrow places" of the program in speed and memory.

7 General Object Features and Object Identification

7.1 Object Levels.

The following requirements have been formulated for the object selection:

- Reusability
- Complete implementation with its own mathematical sense
- Hierarchical structure of objects
- Integrity of objects

According to these requirements all objects were divided into 6 levels. They are described below.

Level 0. Logical operations with float-point and fixed-point numbers. These operations are very close to hardware and they have a simple implementation.

Level 1. Arithmetic operations with float-point and fixed-point numbers. These operations are more complicated then logical operations. Some of them may have different implementations in hardware.

Level 2. Additional functions of float-point and fixed-point numbers. Complex arithmetic.

Level 3. Matrix and vector arithmetic. Sequential elementary signal functions. Basic procedures of signal processing: fft and lattice filtering.

Level 4. Object-oriented operations, which are either standard in the signal processing theory or mathematically beautiful. They do not belong to Level 3 because of their complexity.

Level 5. Specific signal processing functions.

7.2 Interactive Search

There are 4 basic problem statements depending on input data.

1. Input data consists of mathematical formulas or of an other description with the clear semantic sense. Objects are searched in a library of objects by hands by means of specific keywords.

2. Input data is the same but details of the source algorithm are not quite clear. Keywords and structures of keywords are used during the interactive search in the library. A search of an object with a given template is added.

3. The library of objects is not full enough. New objects are to be selected according to special rules for object identification.

4. No information is given except a C-program without any semantic comments. Then new objects are to be selected by the syntax/semantic analysis of the C-code. The rules for selection are formulated below.

The problem statements mentioned depend on existence of an object library and on semantic information that could be used by experts.

7.3 Statistical and Information Approaches and Corresponding Rules

The statistical group of criteria for object selection is based on the "past" algorithms counting their frequency of use and on the "future" algorithms supplied by experts in the fixed research field. A new object is determined if it is statistically significant, that is a big number of similar blocks appear in a fixed research field. This leads to the rule

R1. Similar parts of programs form a single object.

Such parts are to be replaced by a new object. This transformation is worth to do if the new program is shorter and clearer. A mathematical assertion is called theorem if its formulation is easier and shorter than its proof. Hence the following condition should be held.

C1. A length of input and output parameters, which must be determined whenever the object is called, is less than the length of a body of the object.

Any criteria of object selection contain constraints that are defined by special thresholds in the discriminant rules. Profiling of the program gives information about memory and speed of particular segments.

C2. Thresholds for the new object selection are to be weakened if the block represents the "narrow place" of the program.

If no statistics is available then it can be replaced by the experts' opinion. Experts study the reusability feature of the block and predict its expected importance.

The information approach is connected with the notion of a semantic sense, which can be formalized in terms of information flows, or can appear in the interactive dialog between experts and the object identification system.

If an expert is involved in the object selection then he can extract blocks in the program, which have a semantic sense. The next rule can be implemented in the dialog mode.

R2. An expert looks through the program and extracts blocks using vocabulary of keywords that restrict the research area. The final decision of getting a new object in the library is made by the system.

Automatic analysis of the program is the most general tool for object selection. It is based on syntax restrictions of the language and on the information flows between segments of the program.

R3. A block is extracted in the program in accordance with syntax criteria of its information importance.

This means, first, that the block must be admissible, or that its replacement by a function call is not a syntax error. Secondly, the block must be informationally important, that is, the number of inner variables and of operators must be significant comparing with the number of inputs and outputs.