BNF学習支援機能を持つプログラミング環境NBの実現

橋本 裕[†] 早川栄一[††] 並木美太郎[†]
吉澤康文[†] 高橋延匡[††]

プログラミング学習時には，言語文法や，プログラミング技術のほかに，コンピュータサイエンスの諸概念を学習する必要がある．その1つにBNF（Backus-Naur Form）がある．BNFは言語の構文を記述する際に用いられるメタ言語であり，プログラミング言語の構文の記述にも頻繁に使用される．BNFの学習には，(1) 読み方の教育，(2) 書き方の教育，そして動機付けが必要である．BNFの教育にコンピュータを活用できないかと考え，我々は，プログラミング環境NBを設計，実現した．環境NBでは，対話的な可視化と実例主義が重要と考え，現実のプログラミング言語を利用し，BNFの学習支援機構を言語処理系に統合する方法が実現されている．学習を支援するため，次にあげる機能を実現した．(a) 導出過程可視化機能，(b) 構文調査機能，(c) 構文エラー調査支援機能，(d) BNF一覧機能，(e) 言語文法切替え機能．

Implementation of Programming Environment NB with BNF Learning Support Functions

Yutaka Hashimoto,[†] Eiichi Hayakawa,[††] Mitarou Namiki,[†]
Yasufumi Yoshizawa[†] and Nobumasa Takahashi[††]

Programming novices learn brief language syntax and the related programming techniques. Additionally, they should learn computer science concepts. BNF (Backus-Naur Form) is one of the important concepts. It is a meta language for describing language syntax and used frequently to write real programming language syntax. To understand BNF, they should learn how to read it and write it. Because BNF has abstract concepts, motivation is also import not to give up learning. We thought to apply computers to the education, and designed and implemented programming environment NB. Our key concepts are concrete examples and interactive visualization. Therefore, we decided to use real programming languages for the concrete examples and to integrate interactive visual BNF learning support functions to the environment. It has following functions: to visualize the derivation path, to investigate the syntax, to support investigating syntax errors, to visualize a list of BNFs and to change other language syntax.

1. はじめに

プログラミングの学習者が学習すべきコンピュータサイエンスの諸概念はいくつか存在するが，その1つにBNF（Backus-Naur Form）[1)]がある．BNFは言語の構文を記述する際に用いられるメタ言語であり，プログラミング言語の構文の記述にも頻繁に使用される．また，BNFを抽象化の一手法として学ぶこともできる．

この学習に必要と思われる教育内容に以下の項目がある．

(1) 知識の教育

(a) 読み方の教育

BNFをどのように読み，結果どのような文字列が受理されるかを認識させる教育が必要である．

(b) 書き方の教育

読めるだけではなく，自分で，ある文字列群を抽象化してBNFとして表現できるような能力を持たせる教育が必要である．

(2) 動機付け

BNFが対象とする分野は一般的には知られていない．その結果，学習の必要性を理解できない．したがって学習の動機が薄くなる．そこで，何らかの学習の動機付けが必要である．

従来，これらの教育は，黒板などで説明し，その後

† 東京農工大学工学部
Faculty of Engineering, Tokyo University of Agriculture and Technology
†† 拓殖大学工学部
Faculty of Engineering, Takushoku University

教師が生徒に紙面上で問題を解かせるなどをして指導する必要があった．しかし，生徒の側でも教師が示してくれた以上の例を見て理解したいという要求もあった．そのようなことから，これらの教育には，従来の紙面上での教育に加え，コンピュータを用いた教育も有効であると考え，我々は,「プログラミング環境 NB」を設計・実現した．NB は，本学の教育で使用されていたプログラミング言語 Full BASIC[2] が実行可能な統合環境であり，BNF の学習支援機能を持つ．主な特徴は，導出過程可視化機能，構文調査機能，構文エラー調査支援機能，BNF 一覧機能，言語文法切替え機能などである．以降，2 章では BNF の教育支援方針について，3 章では環境の設計について，4 章でその実現と評価について，5 章で，他の関連する研究について，そして 6 章でまとめについて述べる．

2. BNF 教育支援方針

BNF 教育支援に関して 2 つの方針を持つことにした．対話的な可視化と実例主義である．

(1) 対話的な可視化

BNF の読み方を教育するうえで必要となるのは，BNF 自体の構成を簡単に説明し，BNF を解釈していく過程を説明することである．しかし，これだけでは理解が困難と思われるので，実際にその記号列の導出過程を示し理解を深めさせる必要がある．この部分にコンピュータを利用する．そこで，それらを可視化し，コンピュータの特性を生かして対話機能を持たせることで，より深く読み方を学習させることができると考えられる．

(2) 実例主義

初めから，示された文字列群を表現する BNF を記述させるのは難しい．そこで，ある文字列を導出する BNF は，以前見たこのような表記方法だったということを思い出させ，それらを応用していく形式にすると BNF への記述が容易になると考えられる．そこで，よく使われる BNF を学習者に覚えてもらうのが BNF を記述するために必要な第 1 ステップであると考えられる．

同時に，現実に利用されている BNF で学習できれば，その BNF に興味を持つことができ，動機付けとなると考えられる．

3. 設　　計

3.1 設計方針

(1) 現実のプログラミング言語を利用する

現実に利用されているプログラミング言語を採用する．BNF で記述されている言語で現実に使われているプログラミング言語を例とするのが適切であると考えられる．

(2) 言語処理系に統合する

プログラミング環境の提供方法には，BNF 教育専用の環境を提供する方法がある．この方法は BNF の勉強時にしか利用されないので，BNF のフレーズなどが覚えにくいという問題がある．

現実のプログラミング言語を使用する特徴を生かし，その言語を実行できる環境とともに提供する方法がある．この方法の利点は，実際に利用するプログラミング言語なので，興味を持たせやすく，構文エラー時には BNF を調べ，その原因を調査するというような BNF の教育と絡めることが可能な点である．

3.2 BNF 記述方法の設計

一般的には BNF という言葉自体はあいまいに使われている．あるときは，プログラミング言語 Algol60[1] の構文規則の表記法として提案されたような，単純な定義方法だけの表記方法（以降純 BNF）をさし，またあるときは，繰返しや省略といった正規右辺文法を表現可能な EBNF（Extended BNF）をさしている．

純 BNF は単純な定義だけなので BNF 自体の理解は簡単であるが，繰返しや省略などはそれを表現するためだけの BNF 規則が必要となる．実際の言語に適用することを考えると，繰返しや省略のために導入される BNF が増え，そのために構文が増大し，理解しにくくなるという問題が考えられる．

それに対して EBNF では専用の表記方法があるので，直感的に理解しやすいと考えられる．また，構文中に前述のような BNF が含まれなくなることから，見かけ上の構文規則数を減らすことができ，学習者に対するプレッシャーを軽減できる．そして,「学習者に初めに提供するのは分かりやすい表記がよい」という観点から EBNF を採用することにする．

具体的な EBNF の表記方法は決まっておらず，その表記方法は言語作者によってまちまちであることが多い（**表 1**）．そのような状況の中で，本環境では正規表現にも応用可能な表現をとることにした．ただ，本環境が当初対象としていた Full BASIC では ( ) は

表 1 さまざまな EBNF と正規表現
Table 1 Various EBNFs and regular expressions.

	繰返し	省略	グループ化
本環境	*	?	{ }
正規表現	*	?	( )
Full BASIC	*	?	{ }
Ada	{ }	[ ]	{ } [ ]

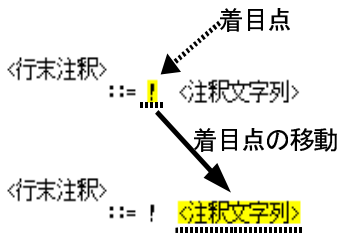

図 1 着目点の動作
Fig. 1 Transition of an attention point.

使用されているが，{ } は使用されていない．メタ記号としての ( ) と言語構文中の ( ) を混同させないようにするために，本環境ではグループ化の記号として { } を採用することにした．

3.3 機 能 設 計

3.3.1 導出過程可視化機能

BNF の理解を手助けするために可視化を行う．そのために可視化すべきこととして次項があげられる．

(1) BNF の読み方の提示

BNF の読み方を教育するために，各 BNF の要素の読み方を可視化する．この様子を観察することで読み方の学習ができる．

(2) 導出過程の提示

BNF の導出過程とそれが導出されるまでの履歴を提示する．各 BNF の局所的な読み方だけではなく，それが今までにどのように全体としてつながってきているのか，という BNF 間の関係を理解することが構文の理解には必要である．

そこで，次のような可視化を行う．

(1) 着目点を利用した可視化

(a) 単純遷移可視化機能

各 BNF の要素の読み方を教育するために，各要素の参照順序を指摘していく必要がある．そこで，着目点という，現在どの部分に着目しているのかを示す点を用意する（**図 1**）．この点を，読んでいく順番に移動させていくことで，その読み方を示す．着目点は，通常は右 1 つとなりの要素へ動く．

(b) メタ記号遷移可視化機能

EBNF では正規右辺文法が記述できるため，繰返しや省略の記述がメタ記号として可能である．したがって，メタ記号の修飾がかかった定義右辺要素に前述の方法を単純に適用すると，突然ほかの非終端記号や終端記号に着目点が移動するように見える．これは，学習者としては不可解に思える．そのようなことから，メタ記号遷移に関する，他の何らかの補助的な可視化方式が必要になると考える．

その 1 つの方法として，単純に左から右へ移動させるが，メタ記号に修飾されている各 BNF の要素があった場合，着目点の状態を変化させ，そこが普通の部分ではないことを示すという方法がある．しかし，この方法ではメタ記号修飾されている部分が階層になっていると，どの記号の影響でなぜそうなるのかが分かりにくいという問題がある．

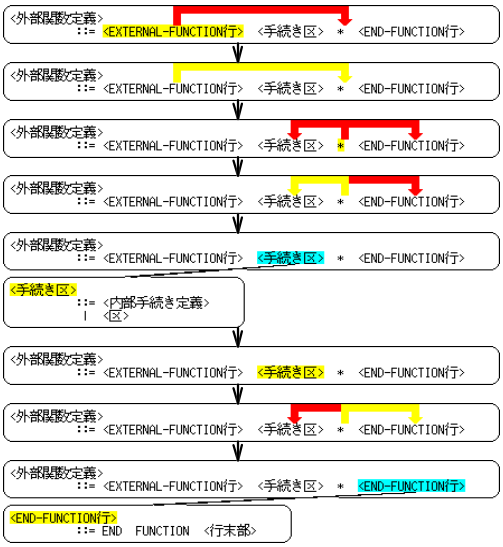

図 2 メタ記号に対する遷移の可視化
Fig. 2 Transition visualization on a meta symbol.

他の方法として，そのメタ記号が表す内容を直接的に表現するという方法がある．たとえば，単純に右に移動しない場合には，それを示す矢印を提示するという方法である（**図 2**）．この方法では，なぜ単純に右に移動しないかという理由が理解できる．そのようなことから本処理系ではこのようなメタ記号遷移可視化方式を採用する．

(2) 導出過程可視化

構文解析では，BNF 右辺にある終端記号や非終端記号の並びに，入力字句の並びが一致しているか否かを検査する．終端記号の場合には，そのまま一致検査を行うが，非終端記号の場合，それが定義している右辺の並びを新たに導出して，それを検査する．BNF の理解にはこれを可視化することが重要である．それを表現するための方法として，文献 3) に述べられている方法がある．この方法では，現在可視化されている BNF と同じ位置に上に新しく導出された BNF を可視化している．この方式は実装は容易であるが，そこまでに導出された過程が理解しにくいという問題があった．本方式ではこの欠点を解決するため，**図 3** に示す方式で可視化することにした．着目点が非終端記号上に来たときに，その記号を開始点とする吹き出しを作成

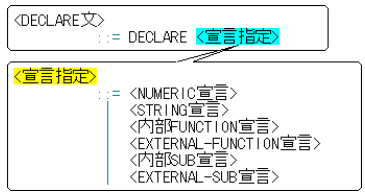

図 3 展開の様子
Fig. 3 An expansion.

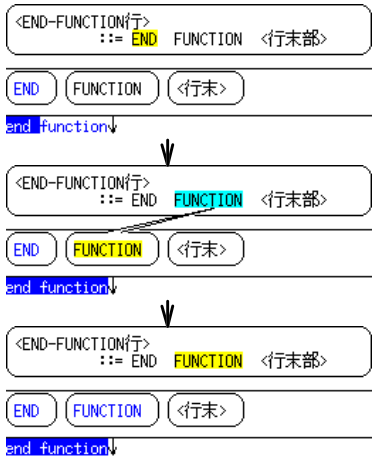

図 4 対応関係の可視化
Fig. 4 Visualization for a token and the related BNF terminal symbol.

図 5 構文エラー時の画面
Fig. 5 A snapshot at a syntax error.

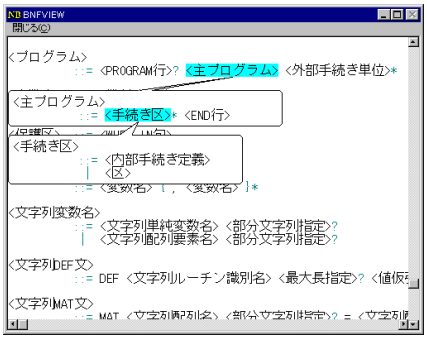

図 6 構文一覧
Fig. 6 A list of BNFs.

し，その中に導出された定義を可視化する．この可視化により，導出過程も知ることができる．
また，着目点が終端記号上に来たときには，ソースプログラム中の特定のトークンとの一致を示している．本環境ではそれを示すためにソースプログラムに対応するトークンへの吹き出しを作成し，着目点をそこに移動させることで，導出の関係を統一的に可視化することにした（**図 4**）．さらに，それがソースプログラム中のどの部分と一致するかを示すため，一致した個所を参照済みの色に変換し，それを示す．

3.3.2 構文調査機能

導出過程を調べているときに，そこで定義されている非終端記号の内容を知りたくなることがある．その場合，一覧からその構文定義を探し出すより，その場で対話的に知ることができれば便利である．そのようなことから，非終端記号をクリックするとその定義内容を図 3 のように展開可視化する対話的な構文調査機能を設ける．当然，その展開された定義内容の非終端記号に対しても調査が可能である．これを繰り返すことで具体的な導出のイメージを得ることができると考えられる．

3.3.3 構文エラー調査支援機能

本処理系では構文エラーも学習のために利用する．構文エラーは，言語の構文と一致しない場合に発生するエラーである．そこで，その構文と不一致である個所の BNF を調査することで原因を理解できる．ただし，調査してもなぜそのエラーが発生したのかがよく分からない場合も存在する．そのようなことから，コンピュータがエラーと判断した理由を示す必要がある．そこで先読み記号を対話的に可視化する機能を提供する（**図 5**）．図 5 の右下のウィンドウで提示された先読み記号をクリックすると，それを導出する構文までの導出経路が左上のウィンドウに可視化される．

3.3.4 BNF 一覧機能

現在の言語での構文規則数の規模を知りたい場合，現在使用されている構文規則をすべて表示する機能が必要となる．そこで，言語構文の BNF 一覧機能を提供する（**図 6**）．構文の配置順序として 2 つの方法が考えられる．

(1) 開始記号から近い順番

その言語の開始記号から始めて，その中で使われている規則順に並べるというやり方である．この場合，並びの上位から閲覧するだけで構文の大体の展開順序が

理解できるような気がするが，現実的にはそのように配置するのは難しい．また，構文規則を名前から探すのが難しいという問題がある．

(2) アルファベット順

構文規則を名前から探すのは上記の方法に比べれば容易である．ただし，単純に閲覧するだけではその展開順序を予測するのは難しい．しかし，ここで前述の構文調査機能が使えるようになっていれば，その構文規則から非終端記号をクリックするだけでその定義内容を調べることができ，展開順序を知ることができる．

このようなことから，アルファベット順に示すことにする．また，並べる順番はアルファベット順でその次に日本語での定義を並べる．探しやすくするため，日本語はコード順ではなく，読み順で並べる．

3.3.5 言語文法切替え機能

プログラミング言語 Full BASIC では構文規則数は約 500 あり，決して少なくはない．そのようなことから，これらすべての構文を学習者に一度に提示すると混乱する可能性がある．そこで少しずつ使える構文を増やして提供できる機能の提供を考えた．たとえば，簡単な入出力と関数の引用といった程度の構文だけにすると，約 140 の構文規則ですむ．本機能では，使用する言語文法を複数切り替える機能を提供する．それぞれの言語のサブセット言語を使用することで，学習段階に応じた言語とその構文規則を提供できる．

4. 実現と評価

4.1 システム構成の設計指針

- 統合環境の提供
  現実に存在する言語を利用し言語処理系に統合するという要求がある．つまり，ソースプログラムを記述し，それを実行可能形式に変換し，実行する環境が必要となる．それぞれを個別のツールとして提供するより，1 つにまとめて提供したほうが，利用者の操作の学習の手間を軽減できるうえに，BNF 以外のコンピュータサイエンスの基礎概念の学習支援機構も実現しやすくなる可能性がある．そのようなことから，編集や，変換，実行を 1 つの環境で行える統合環境で実現をする．そして，前述の設計から，グラフィックスを扱え，かつポインティングデバイスを使えるウィンドウシステム上で実現する．この結果として，実行結果を見ながら，ソースプログラムを編集するということが可能になる．
- 個別ウィンドウでの提供
  実際にプログラムを実行するには，ソースプログラムの編集，実行可能形式の変換，その実行という各段階が必要であり省略はできない．したがって，編集と実行系への変換をまとめたりせずに，そのまま各段階を提供することが，学習者の，実行への段階の学習の認識によいと考えた．そのようなことから，各段階それぞれに個別のウィンドウを割り当て，各機能を提供する（編集系，変換系，実行系）．BNF の学習支援は変換系で行うことになる．また，操作の便宜を図るために別途操作専用の制御系のウィンドウを用意した（**図 7**）．

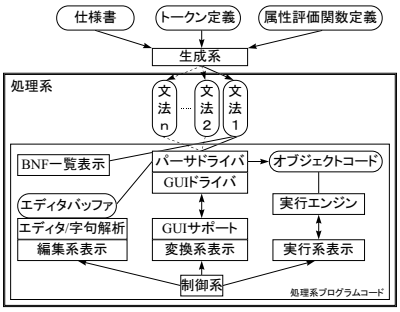

図 7 システム構成
Fig. 7 System structure.

- トップダウンな解析方法の採用
  構文の解析方法にはトップダウンな解析方法とボトムアップな解析方法が存在する．トップダウンな解析は，ある 1 つの BNF から導出させていくことができる．それに対して，ボトムアップな解析では，複数の記号列や非終端記号が BNF へ変換されていくように見える．本方式での可視化は導出過程を見せることが要求されているので，トップダウンな解析方法が適切である．正規右辺文法をトップダウンな解析で可視化する方式として，文献 4) がある．本処理系ではこれを採用することにした．文献 4) の方式では，仕様書を与えると解析に必要なデータを出力する．そこで，あらかじめ複数の構文セットのデータを作成しておき，解析時に使用するデータを切り替えることで，言語文法切替え機能を提供する．

4.2 実現環境と規模

実現は，MS-DOS 上に主にプログラミング言語 C を用いて実現した．ソースの規模を**表 2** に示す．本環境では，言語文法切替え機能がある．そのために LL(k) パーサ生成系を作成した．その出力はリソースであり，それを最後にオブジェクトモジュールにリソースバインダにより結合する．

MS-DOS 上でグラフィックスを統一的に扱うために，オーバラップが可能なウィンドウシステムを実現

表 2 ソースファイルの規模
Table 2 Lines of source files.

名前	行数
本環境	22,847
マルチスレッドライブラリ	1,020
ウィンドウシステム	8,581
LL(k) パーサ生成系	3,520
リソースコンパイラ	980
リソースバインダ	104
合計	37,052

表 3 アンケートの結果
Table 3 Result of questionnaires.

有用度	5 段階平均
BNF 学習	3.6
Full BASIC 構文理解	3.1
メタ記号遷移可視化機能	4.4
構文調査機能	4.8
構文エラー調査支援機能	4.0

し，そのためにマルチスレッドライブラリを実現した．本環境はこのウィンドウシステム上で動作する．

後に本環境は MS Windows 上に移植された．GUI とウィンドウを使ったアプリケーションにもかかわらず，2,000 行程度の修正により移植することができた．移植は主にウィンドウシステムのエミュレーションルーチンを作成することにより行われた．修正行数の 75%はこのエミュレーションのコードである．

4.3 評　　価

Full BASIC プログラミング経験のある本学の学部生 5 人に，処理系の有用度に関する知見を得るための評価を行った．まず初めに処理系の簡単な使い方を説明したあと，構文エラーを持つプログラムを提示し，処理系を使用して，なぜエラーになるのかを BNF を提示して説明させるなどの問題を与えた．時間は 1 時間とした．そのあと，処理系の有用性に関する 5 段階評価のアンケートをした．アンケートの回答を**表 3** に示す．

まず，BNF 学習と Full BASIC 構文理解に関する有用度では，有益であると感じていることが分かった．しかし Full BASIC 構文理解のほうが多少値が低い．これは，短期の実験だったので，Full BASIC の構文として理解する必要に迫られず，構文に関した問題というよりも，単に BNF に関する問題であるととらえられたからではないかと考えられる．また，各機能に対しては，おおむね有用であると感じてもらえたようだ．特に，メタ記号遷移可視化機能と構文調査機能に関しては，流れを追いやすく有用であるとのコメントを得た．

結論として，今回の結果からは，BNF の学習のための各機能に関しては有用と感じており，また BNF の学習や Full BASIC の構文理解の観点からも有用であるという評価を得た．

5. 関 連 研 究

コンパイラの学習支援としては以下にあげるものがあるが，編集・変換・実行，BNF 学習が可能な統合プログラミング環境ではなく，また解析の様子の可視化は本環境が提供する可視化手法と異なっている．

(1) 手書きによる再帰下向き構文解析[3)]
このプログラムは，プログラミング言語 C のサブセット言語を対象に，構文解析やシンボルテーブルといった部分の可視化を行っている．構文解析の可視化では構文図式による可視化を行っており，非終端記号の展開時にはその表示部分を置き換えるような形で，展開されたものを表示する．

(2) Gyacc[5)]
Yacc を用いて解析の表やスタック状態，導出木を可視化する．正規右辺文法は使用できず，構文解析自体を BNF を用いて可視化するようなことも行えない．

(3) Bison を用いた方法[6)]
Bison を用いて作成された output ファイルと，その生成されたパーサを使用して取得するログを元に，ソース中の先読み記号，スタック，LR 状態，還元時に適用された生成規則，解析木を可視化する．正規右辺文法は使用できない．

(4) LLParse，LRParse[7)]
小規模の入力を対象としたツールで，表や解析スタックの状態を見ることができる．

(5) コンパイラ教育システム[8)]
LL(1) と SLR(1) の解析が可能で，適用された生成規則，それまで適用された規則，構文図式，構文木などの情報を可視化できるが，利用者が非終端記号の内容を指定して見たりすることはできないようであり，それまでの適用された規則も単なるログのような表示になっているので関係が見にくいという問題がある．

統合環境型の学習環境はいくつか存在しているが，これらはその環境を BNF の学習支援に使用できない．

(1) THETIS[9)]
プログラミング言語 C を実行できる統合環境で，具体的な型名や変数名を使ったエラーメッセージを出力できたり，間違いやすい文法を制限したりしている．また現在の実行位置や変数の値を見ることができる．

(2) PASCAL/V[10)]
プログラミング言語 PASCAL を対象に，実行時の配

列の内容や現在の実行位置を知ることができる．また，配列に違う型の値を代入しようとしたエラーは単純にエラーメッセージを表示し，配列の添え字が範囲外になったときのエラーはそのエラー場所を指摘することができる．

(3) GRASP [11)]

この環境は，制御構造の可視化である制御構造図と，文レベルの複雑性の可視化である複雑性輪郭グラフの自動生成ができ，そのうえで 1 種類の言語だけではなく，複数の言語を同じ環境で使えることを目指している環境である．

6. おわりに

プログラミングの学習者が学習すべきコンピュータサイエンスの諸概念の 1 つである BNF に関して，読み方の教育，書き方の教育，動機付けを行えるような教育用 Full BASIC プログラミング統合環境の設計と実現を行った．それらの教育には，対話的な可視化と実例主義が重要と考え，現実のプログラミング言語を利用し，学習支援機能を言語処理系に統合する方法を提案した．主な特徴は，導出過程可視化機能，構文調査機能，構文エラー調査支援機能，BNF 一覧機能，言語文法切替え機能を持つことにある．

参 考 文 献

1) Naur, P., Backus, J.W., Katz, C., Rutishauser, H., Wegstein, J.H., Bauer, F.L., McCarthy, J., Samelson, K., van Wijngaarden, A., Green, J., Perlis, A.J., Vauquois, B. and Woodger, M.: Reviced Report on the Algorithmic Language ALGOL 60, *Comm. ACM*, Vol.6, No.1, pp.1–17 (1963).
2) 西村恕彦，植村俊亮，酒井俊夫，高田正之（編）：アメリカ規格 Full BASIC，共立出版 (1990).
3) 岩崎克治，辻野嘉宏，都倉信樹：教育を目的としたコンパイラの動作の可視化手法とその効果について，情報処理学会コンピュータと教育研究会，Vol.90, No.39, pp.17–24 (1990).
4) 橋本 裕，早川栄一，並木美太郎，吉澤康文，高橋延匡：正規右辺文法に対応した位置番号に基づく LL 構文解析可視化アルゴリズム，電子情報通信学会論文誌，Vol.J84-D-I, No.4, pp.326–387 (2001).
5) Lovato, M.E. and Kleyn, M.F.: Parser Visualizations for Developing Grammars with Yacc, *SIGCSE Bulletin*, Vol.27, pp.345–349 (1995).
6) 楠目勝利，佐々政孝：コンパイラにおける構文解析過程の視覚化，情報処理学会第 55 回全国大会，pp.1-448–1-449 (1997).
7) Blythe, S.A., James, M.C. and Rodger, S.H.: LLparse and LRparse: Visual and Interactive Tools for Parsing, *SIGCSE Bulletin*, Vol.26, No.1, pp.208–212 (1994).
8) 中西健一郎，梅田雅之，和田幸一，川口喜三男：アルゴリズムの可視化に基づくコンパイラ教育支援システム，情報処理学会第 48 回全国大会，pp.1-5–1-6 (1994).
9) Freund, S.N. and Roberts, E.S.: THETIS: An ANSI C Programming Environment Designed for Introductory Use, *SIGCSE Technical Symposium*, Vol.28, No.1, pp.300–304 (1996).
10) BRETTE, J.-F.: Transparent running and contextual help to learn and to teach an imperative language, *SIGCSE Bulletin*, Vol.27, No.2, pp.7–12 (1996).
11) Hendrix, T.D., II, J.H.C., Barowski, L. and Teate, J.C.: A Visual Development Environment for Multi-Lingual Curricula, *SIGCSE Bulletin*, Vol.29, No.1, pp.20–24 (1997).

(平成 14 年 4 月 1 日受付)
(平成 14 年 11 月 5 日採録)

橋本 裕（正会員）

1994 年東京農工大学工学部電子情報工学科卒業．1996 年同大学大学院電子情報工学専攻博士前期課程修了．同年同大学院電子情報工学専攻博士後期課程進学．2002 年同大学院電子情報工学専攻博士後期課程単位取得退学．プログラミング言語処理，およびその開発環境に興味を持つ．

早川 栄一（正会員）

1989 年東京農工大学工学部数理情報工学科卒業．1992 年同大学大学院電子情報工学専攻博士後期課程単位取得退学．同年同大学電子情報工学科助手．1998 年拓殖大学工学部助手．博士（工学）．1999 年拓殖大学工学部専任講師．オペレーティングシステムをはじめとするシステムソフトウェアの研究に従事．ACM，IEEE，電子情報通信学会各会員．

並木美太郎（正会員）

1984 年東京農工大学工学部数理情報工学科卒業．1986 年同大学大学院修士課程修了．同年 4 月（株）日立製作所基礎研究所入社．1988 年東京農工大学工学部数理情報工学科助手，1989 年電子情報工学科助手，1993 年 11 月電子情報工学科助教授，1998 年 4 月情報コミュニケーション工学科助教授．博士（工学）．オペレーティングシステム，プログラミング言語，ウィンドウシステム，並列処理，コンピュータネットワークおよび日本語情報処理の研究・開発に従事．ACM，IEEE，電子情報通信学会各会員．

吉澤　康文（正会員）

1967 年東京工業大学卒業．同年（株）日立製作所・中央研究所に勤務．1973 年同社システム開発研究所に転勤．この間，仮想記憶，大規模 TSS，オンラインシステム等，大型計算機の OS 研究，開発ならびに性能評価の研究に従事．また，OS のテスト・デバッギングシステムの開発，ハイエンドサーバ，超並列計算機，リアルタイムシステム等の研究開発に従事．1995 年 10 月東京農工大学教授．工学博士．情報処理学会論文賞（1972 年）．情報処理学会フェロー会員．ACM，IEEE/CS 会員．現在，メディア情報処理，モバイルコンピューティング等に興味あり．

高橋　延匡（正会員）

1957 年早稲田大学第一理工学部数学科卒業．同年 4 月（株）日立製作所中央研究所入社，HITAC5020 モニタ，TSS の開発等に従事．1977 年東京農工大学工学部数理情報工学科教授．1989 年電子情報工学科教授．1997 年拓殖大学工学部教授．2002 年 6 月没．オペレーティングシステム，日本語情報処理等の研究，教育に従事．理学博士．