プログラミング言語としての日本語

岡田　健[†1]　中鉢欣秀[†2]　鈴木 弘[†3]　大岩 元[†4]
[†1]慶應義塾大学政策・メディア研究科　[†2]慶應義塾大学ＳＦＣ研究所
[†3]東京都立航空工業高等専門学校　[†4]慶應義塾大学環境情報学部

概要

正しく表記された日本語を用いて記述できるプログラミング言語「言霊」を設計・実装した。正しい日本語が使用できることや具象文法の設定を変更できることなど、可読性の高いプログラムを記述できるよう言語を設計した。日本語をプログラミング言語として用いると文法教育を必要としない為、日本人に対するプログラミング入門教育ではプログラムの表現能力の育成に集中できる。また「言霊」は **Java** の実行環境である仮想計算機のアセンブラを日本語化し、低級な機械語表現から高級表現まで日本語を用いてシームレスに表現できる。機械語レベルの細かい処理を記述できる為、熟練プログラマのための言語としても有効性が期待できる。

１．はじめに

プログラミング教育とは、自分の考えを論理的にプログラムで表現する能力を育成する事である。その事を通じてコンピュータがどんなものかを理解して、自分の考えたものがコンピュータ上で実現できるかどうかを判断できるようにするという、教養教育である。

だが現在行われているプログラミング教育の多くはコンピュータを使えるようになる為の実用教育から来たものが多い為、表現能力を育成する教養教育の観点からは問題があるものが多い。プログラムが書けるようになることを重視する為、文法教育と定型パターンの利用で実用目的が達成され、そこで終わってしまう。これではパターンで表せない自分の考えを表現できるようにならない。

筆者は慶応義塾大学環境情報学部大岩元教授の元でプログラミング教育をいくつか手がけた。エクサ社における新入社員教育カリキュラムの作成**[1]**や、大岩教授の授業「オブジェクトプログラミング」教材作成などの活動に携わった。

これらプログラミング教育の中で最も重視したのは、プログラムの可読性である。受講者の論理的な思考能力・表現能力の向上の為、やりたい事を如何にプログラムで表現するかを重視した。やりたい事を論理的に理解して表現できているならば、他人が読んでも容易に理解できる可読性の高いプログラムができるはずである。

例えば筆者らが行っているプログラミング入門教育では図１のような **Java** プログラムを教材としているが、ここで重視するのは日本語で書くコメントの存在である。コメントはそれをつけた対象とするプログラム部分の目的を書く。そして、目的の階層構造がプログラムを構成する。即ち、図１のプログラムの目的は家を書く事であり、その目的を達成する為に屋根と家の本体を描く。屋根は三角形で表現し、本体は四角

```
public class MyTurtle extends Turtle{
  public static void main(String args[]){
    house(100); //大きさが 100 の家を描く
  }
  //家を描くプログラム
  public static void house(int size){
    triangle(size); //屋根を描く
    square(size); //本体を描く
  }
  // 長さが size の三角形を描く
  public static void triangle(int size){
    rt(30);   fd(size);
    rt(120);  fd(size);
    rt(120);  fd(size);
    lt(30);
  }
  // 長さが size の四角形を描く
  public static void square(int size){
    for(int i=0 ; i<4 ; i++ ){
      rt(90);
      fd(size);
    }
  }
}
```

図１ 家を描く Java プログラム

形で表現する。

このようにコメントは作ろうとするプログラムの構造を設計する文書になっている。このコメントを具体化するために **Java** で実装するわけである。

しかしここで問題なのは、設計に用いた日本語とプログラムに用いる **Java** の間には、その表現に大きな隔たりがある事である。受講者は **Java** 文法を学んだ上で、思考に用いた日本語から **Java** への翻訳を行う必要がある。だが **Java** 文法教育は上手くいかない事がしばしばあり、やりたい事は日本語で論理的に説明できるしコメントとして記述する事もできるが、それを **Java** プログラムで記述する事ができないという事が多々起きるのである。

日本語表現をそのままプログラミング言語として扱う試みとして、水谷の「朱唇」

```
メインとは {
  大きさが１００の家を描く。
} ことである。

大きさが「A（整数型)」の家を描くとは {
  長さがAの三角形を描く。//屋根を描く
  長さがAの四角形を描く。//本体を描く
} ことである。

長さが「A（整数型)」の三角形を描くとは {
  右に３０度曲がる。　A歩進む。
  右に１２０度曲がる。A歩進む。
  右に１２０度曲がる。A歩進む。
  左に３０度曲がる。
} ことである。

長さが「A（整数型)」の四角形を描くとは {
   {
    右に９０度曲がる。
    A歩進む。
  } を４回繰り返す。
} ことである。
```

図２ 家を描く日本語プログラム

がある**[2]**。しかし「朱唇」はマニュアルが文語で書かれており、プログラムの表記にカナ文字を主として使うなど、個性的すぎて一般に普及するには至っていない。初心者教育には、**Logo** を日本語化したもの**[3]**も用いられているが、そもそもプログラミングを一般人に教える必要がないと考える人が多く、普及が進んでいない。

筆者らは、受講者が正しい日本語表現を用いて記述できるプログラミング言語として、日本語プログラミング言語「言霊」を開発した。

以下本稿では、２節では「言霊」の概要を解説し、実行環境やソースプログラムを紹介する。３節では、既存の日本語プログラミング言語と比較する事で、「言霊」が正しい日本語表記を重視している事を述べる。４節では、「言霊」の解析手法が既存のプログラミング言語とは異なり非決定性オートマトンを使う事や、字句解析と構文解析を

```
if( x==0 )
  y=1;
else
  y=2;
```

図３ Java 言語の記述

```
  iload_1
  ifne Label_0
  iconst_1
  istore_2
  goto Label_1
 Label_0:
  iconst_2
  istore_2
 Label_1:
```

図４ アセンブラ Jasmin による記述

同時並行的に行っている事などを述べる。５節では、「言霊」の手続き宣言と手続き呼び出しの記述を紹介し、既存のプログラミング言語と比べて可読性が向上していることを述べる。６節では、「言霊」が抽象文法と具象文法の考え方を取り入れていて、具象文法は自由に変更できる事を述べる。７節では、「言霊」が低級表現と高級表現を混在させる事により、熟練プログラマにとっても有用な言語である事を述べる。最後に８節でまとめを行う。

２．日本語プログラミング言語「言霊」

筆者は日本語とプログラミング言語の間の隔たりを埋め、正しい日本語で自由に記述できるプログラミング言語「言霊」を開発した。図２は、タートルグラフィックスを用いて家を描くプログラムを「言霊」で記述した例であり、図１と意味的に同じ内容である。このように正しく記述された日本語は、そのままプログラミング言語として機能する。

「言霊」で作成したプログラムは、Java の環境を利用する。ソースプログラムは最

```
  ローカル変数１番目から整数を積む。
  ０以外ならば、ラベル０にジャンプする。
  整数１を積む。
  ローカル変数２番目に整数を格納する。
  ラベル１にジャンプする。
ラベル０:
  整数２を積む。
  ローカル変数２番目に整数を格納する。
ラベル１:
```

図５　日本語による低級表現

```
x が０ならば、１を y に代入する。
そうでなければ、２を y に代入する。
```

図６　日本語による高級表現

終的に Java のクラスファイルに変換され、JavaVM 環境で実行される。その為、既存の JavaAPI などのライブラリは「言霊」上で利用可能である。

現在広く利用されている Java 言語は、実行する際に仮想計算機（JavaVM）を用いる。Java プログラムを記述すると、コンパイラは JavaVM が解釈できる機械語に変換する。その機械語を JavaVM が解釈して演算する事で Java プログラムは実行される。例えば図３のような Java 言語のコードがあったとする。Java コンパイラはこれを機械語に変換するが、その機械語を Jasmin[4]と呼ばれるアセンブラのニモニックで表現したものが図４である。実行時には図４のニモニックで表された機械語命令に従って演算が行われる。

「言霊」は、Jasmin 形式のニモニックを日本語化し、かつその表現を拡張する方向で設計された。例えば図３・図４を「言霊」で記述すると、図５のような機械語レベルの日本語表現になる。だが図５のような記述は低級表現であり表現が複雑になるので、図６のような高級表現も可能にしている。

```
カメ太＝タートル！作る。
「カメ太！１００　歩く　９０　右回り。」
！４かい　くりかえす。
```

図７　ドリトルによる四角形を描く タートルプログラムの記述

```
カメ太（タートル型）を生成する。
 {
  カメ太が１００歩進む。
  カメ太が右に９０度曲がる。
} を４回繰り返す。
```

図８「言霊」による四角形を描くタートルプログラムの記述

```
４を　回数指定し
        １００歩　前進し
        ９０度　右を向き
繰り返す
```

図９ MIND による四角形プログラム

```
 {
    １００歩進む。
    右に９０度曲がる。
} を４回繰り返す。
```

図 10　「言霊」による四角形プログラム

３．「言霊」と他言語の比較

「言霊」の特徴は、正しい日本語表記を重視している事だ。これは他の日本語プログラミング言語と比較する事で分かる。図７は学校教育を目的として開発されたドリトル[5]であり、この例はタートルグラフィックスを使って四角形を描くプログラムである。全く同じプログラムを「言霊」で記述すると、図８のようになる。なおドリトルはオブジェクト指向言語でありタートルもオブジェクトとして扱われるので、図８の「言霊」の表記も同様にタートルオブジェクトを生成するプログラムになっている。「！」のような記号が無い為に「言霊」は自然な表現になっている。

また図９は MIND[6]のタートルグラフィックスを使って同様に四角形を描くプログラムであり、図 10 はそれを「言霊」で記述したものである。MIND は言葉と言葉の間に空白を入れる分かち書きが必要な言語である。例えば

９０度　右を向き

という MIND プログラムは、

９０度　右を　向き

９０度右を　向き

などのように空白を入れる場所を変えると文章の意味が変わってしまう。「言霊」では「右に９０度曲がる」と分かち書きをせずに記述する事が可能である。

以上のように「言霊」ではドリトルや MIND と異なり、分かち書きや特殊な記号などを使わない自然な日本語表現を重視している。

４．「言霊」の自然言語処理

「言霊」におけるコードの解析は、従来のプログラム言語の解析の手法とは異なる。Java など従来の手法での解析は、まず字句解析が終了してから構文解析を行っている。例えば「a=10+20」という Java のコードを解析する場合は、まず字句解析を行って [a][=][10][+][20] のようにコードを字句に分割する。このような字句解析が可能なのは、Java には予約語が存在していて、これにより字句と字句の区切りが判別できるからである。先のコードの場合は「=」「+」が予約語であり、これらの文字は変数名などには使用できないため、字句に分割する事ができる。構文解析は字句解析が終了してから行われる。ここでは字句の中に「=」

があることから、このコードが代入文であ　　事が分かる。

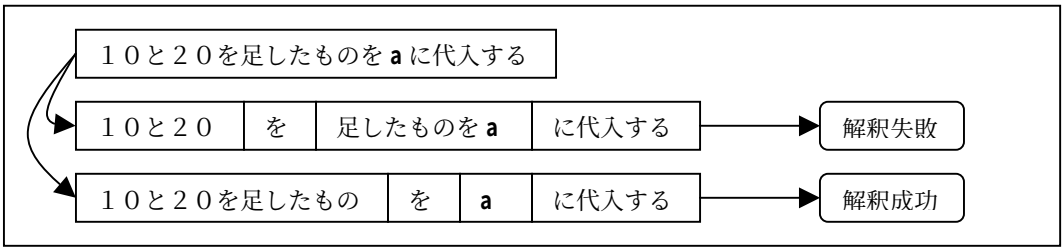

図 11 「言霊」の解析

る事が分かる。

「言霊」はこうした従来の手法とは異なり、字句解析と構文解析を同時並行的に行う必要がある。例えば「１０と２０を足したものを a に代入する」というコードを解析する場合は、以下のような手順を踏む。「言霊」には、代入文と宣言文など数通りの文が存在する。だが解析前はコードがどの種の文なのか分からないため、それぞれの文型にマッチするか判定を試みる。例えば代入文は「(A) を (B) に代入する」という文型をしているので、コードは図 11 のように２通りにマッチする事が分かる。前者は (A) が「１０と２０」、(B) が「足したものを a」、後者は (A) が「１０と２０を足したもの」、(B) が「a」となる。

その上でさらにそれぞれの (A) と (B) の部分の解析を進めると、「１０と２０」や「足したものを a」という表現はこの場所に相応しいものではないことが分かり、前者の解釈は切り捨てられる。その結果 (A) に適切な算術表現が入っていて (B) に適切な変数名が入っている後者の解釈が正しいと判定されて、このコードが代入文である

既存のプログラミング言語の解析は決定性オートマトンを用いて実装されるが、「言霊」コンパイラは非決定性オートマトンを用いて実装されている。既存の言語は予約語が存在している為、受理した文字により状態遷移の方向が確実に決定できる為に決定性オートマトンにより実装できる。

「言霊」の場合は状態遷移の方向を確実に決定する事ができない。図 12 は、「言霊」において代入文を受理する為の非決定性オートマトンである。最初に「を」の文字を受理するまで状態１のまま自己遷移を続け、受理している文字列が変数名として認識される。だが「を」を受理したと言っても、単純に状態２に遷移するわけにはいかない。なぜなら変数名に「を」の文字が使われているかもしれないからだ。そこでコンパイラはオートマトンを二つに分裂させる。片方は状態１のまま、もう片方は状態２に遷移したものを作成し、以後は同時並行的に受理を続ける。誤った受理をしているオートマトンはそのうち行き詰まるので、その時にオートマトンを消滅させる。そして最終的に残ったオートマトンを採用するので

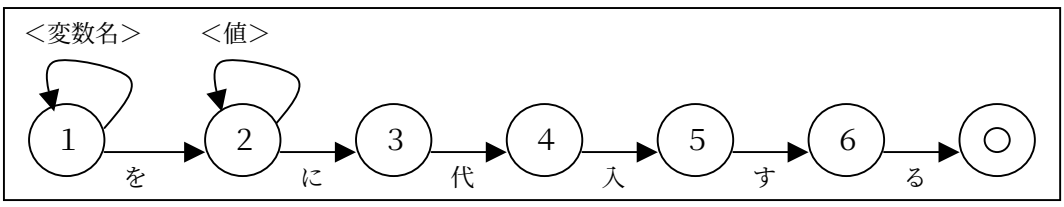

図 12 非決定性オートマトンを用いたコンパイラの実装

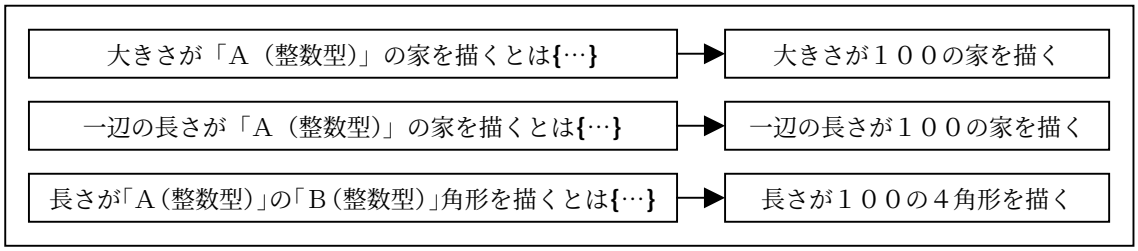

図 13 「言霊」における手続き宣言と手続き呼び出し

ある。図 **12** では単純化してあるが、実際は「値」を受理する為のオートマトンも存在し、そこで同様にオートマトンの分裂と消滅を行っている。

非決定性オートマトンを用いた処理が必要なのは、「言霊」が可読性の高いプログラムを記述できるように予約語の存在を極力排除している為である。既存のプログラム言語は予約語の存在により決定性オートマトンによる解析が可能だが、「言霊」の場合は非決定性オートマトンを用いる必要がある。

以上のように「言霊」のコード解析は従来の手法とは異なり、字句解析と構文解析を平行に行いながら、複数ある可能性を全て探索して正しく解析が行われたものを選び出す。これは自然言語処理で使用されている手法である。

５．「言霊」における手続き

「言霊」で特徴的なのは手続きの宣言と呼び出しの表現である。図２ではタートルを用いて家を描くコード例を示したが、そこで記述されている家を描く手続きをはじめ、「言霊」における手続き宣言と手続き呼び出しの例を図 **13** にあげた。

「言霊」における手続きは文型により宣言されて解釈されるため、既存のプログラム言語よりも可読性の高いコードを記述する事が可能である。図 **13** の手続きは「大きさが～の家を描く」という手続き宣言を行っている。だがこの手続きでは意味が伝わりにくいと思ったら、「一辺の長さが～の家を描く」という手続きに変更する事も可能である。また、複数の引数が存在した場合も柔軟に文型を変更する事は可能である。例えば一辺の長さを引数に多角形を描く手続きを宣言する場合、「～辺を持ち、長さが～の多角形を描く」として「４辺を持ち、長さが１００の多角形を描く」と呼び出すか、あるいは「長さが～の～角形を描く」と宣言して「長さが１００の四角形を描く」とする事も可能である。

既存のプログラム言語において、手続きは名前と引数の数と順序により解釈していた。だが「言霊」では手続きの文型により解釈する事で、**Java** やＣ言語と比較しても可読性の高いプログラム記述を可能にしている。

６．「言霊」の抽象文法と具象文法

「言霊」におけるコード解析は、文法表現設定により決定される。例えば先の例では代入文は「(値）を（変数）に代入する」という文法表現設定である。このように、代入概念を表す文をどのように表現するかを設定すればコンパイラはその通りに解釈する。

この文法表現設定は、ユーザにより自由に変更する事ができる。代入文は「(変数）には（値）を代入する」や「(値）を（変数）とする」と設定する事もできる。これによ

り各ユーザは自分が自然と感じる文法表現設定を行う事で、自分にとって自然なプログラム表現を追及することができる。

文法表現設定を自由にすると各ユーザにより表現がばらばらになる危険性があるが、文法表現設定を元にしたソースの変換を行えば問題は無い。ある人の記述したソースを解析して意味を取り出し、これを他の人の文法表現設定に適用すればよいのである。ある人が記述した代入文が「２０は A である」だったとする。「(値) は (変数) である」という文法表現設定があれば、値が２０であり変数が A である事が分かる。そして他の人の文法表現設定が「(変数) に (値) を入れる」であったなら、「A に２０を入れる」と当てはめる事ができる。このように文法表現設定があれば、ユーザ間の表現の変換を行う事ができる。

文法表現設定を使えば、日本語以外の表現もサポートできるし既存のプログラム言語への変換も可能になる。例えば「A に２０を入れる」という表現は「A=20」になるのである。また「substitution 20 for A」と英語に変換できるし、他言語への変換も可能である。

文法表現設定の考え方の根本には、抽象文法の考えがある[7]。プログラムは抽象文法と具象文法の二つから成り立っている。抽象文法とは「代入文とはどのようなものか」を指す。抽象文法により、代入文は代入対象と代入データを持つ概念であり、代入対象は変数であり、代入データは変数か数値か手続きの返り値である、と言う事を定義する。具象文法は、「代入文をどのように表現するか」を指す。抽象文法によって代入文と言うものがどういうものかが定義されているので、代入対象と代入データさえ持っていれば具象文法でどのような表現を定義しても良い。これまで議論してきた文法表現設定とは、具象文法の定義の事である。そして表現の変換が可能なのは、様々な具象文法を持っていても抽象文法は同じなのだから相互変換が可能であると言う事である。

「言霊」は現在の実装では、Java 言語とほぼ同じ抽象文法を持っている。その為「言霊」と Java 言語は相互に変換が可能である。（もっとも、Java 言語と「言霊」の相互変換には意味が無い。「言霊」の最大の魅力は可読性の高い表現であり、具象文法である。変換を行うとその表現が失われてしまう)。だが将来的には、「言霊」は Java 言語とは異なる抽象文法を持つことになるかもしれない。

図 14 に、「言霊」の抽象文法をＵＭＬのクラス図で表したものを載せる。

７．低級表現と高級表現の混在

「言霊」は低級な機械語命令と高級な文法を混在させる事で、熟練プログラマにとっても有効な言語になっている。「言霊」は JavaVM 環境で動作するため、JavaVM の機械語命令に変換される事は先に述べた。JavaVM はスタックを計算機構として用いていて、これはスタックマシンと呼ばれている。機械語命令はこの JavaVM の実装を理解していなければ使いこなす事ができない為、初心者が扱う類の命令ではない。だが上級者にとっては、機械語命令を使いこなせば JavaVM の能力を限界まで引き出す事が可能になる。「言霊」は、低級表現と高級表現の混在を許す事で、これを可能にしている。

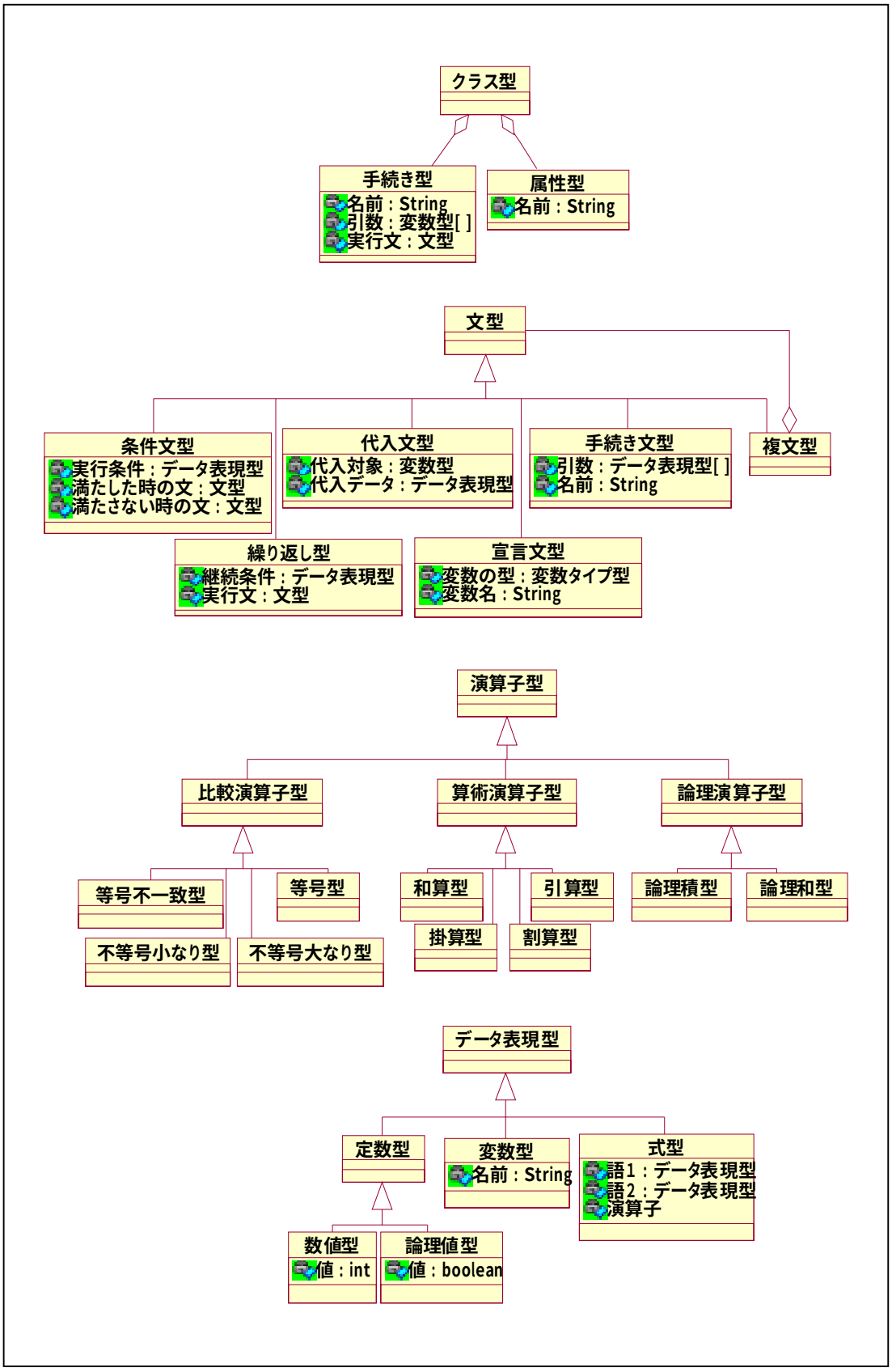
クラス型
手続き型
名前：String
引数：変数型[ ]
実行文：文型
属性型
名前：String
文型
条件文型
実行条件：データ表現型
満たした時の文：文型
満たさない時の文：文型
代入文型
代入対象：変数型
代入データ：データ表現型
手続き文型
引数：データ表現型[ ]
名前：String
複文型
繰り返し型
継続条件：データ表現型
実行文：文型
宣言文型
変数の型：変数タイプ型
変数名：String
演算子型
比較演算子型
算術演算子型
論理演算子型
等号不一致型
等号型
不等号小なり型
不等号大なり型
和算型
引算型
掛算型
割算型
論理積型
論理和型
データ表現型
定数型
変数型
名前：String
式型
語1：データ表現型
語2：データ表現型
演算子
数値型
値：int
論理値型
値：boolean

図 14 「言霊」の抽象文法

```
   ローカル変数１番目から整数を積む。
   ０以外ならば、ラベル０にジャンプする。
   整数１を積む。
   ローカル変数２番目に整数を格納する。
   ラベル１にジャンプする。
ラベル０:
   整数２を積む。
   ローカル変数２番目に整数を格納する。
ラベル１:
```

図５　日本語による低級表現

```
x が０ならば、１を y に代入する。
そうでなければ、２を y に代入する。
```

図６　日本語による高級表現

例をあげる。図５は単純な **IF** 文のプログラムを機械語命令で記述したもので、図６はそれを高級表現に書き換えたものである。「言霊」では、図 **15** のようにこの両者の表現を混在させる事ができる。すなわち分岐に関する表現は機械語命令では煩雑になるので避け、スタック操作に関する命令だけを機械語命令で記述している。こうして混在表現をすると、「ローカル変数２番目に整数を格納する」という命令が重複していることが分かる。そこで図 **15** のように最適化したコードを記述する事ができる。このようにプログラマが低級な機械語命令を扱う事ができる事で、速度と容量で最適なコードを記述する事ができるのである。

８．おわりに

完全な日本語として書かれたものが、プログラムとして **Java** の実行環境で動作するようなプログラミング言語「言霊」について述べた。「言霊」は日本人の初心者にプログラミング教育を行う場合、文法事項について教育する必要がないため、プログラム作成自体に集中できる。従来のプログラミング教育では、言語の文法を修得するこ

```
x が０ならば {
   整数１を積む。
   ローカル変数２番目に整数を格納する。
} そうでなければ {
   整数２を積む。
   ローカル変数２番目に整数を格納する。
}
```

図 15 低級と高級を混ぜた表現

```
x が０ならば {
   整数１を積む。
} そうでなければ {
   整数２を積む。
}
ローカル変数２番目に整数を格納する。
```

図 16 最適化表現

とにほとんどの努力が集中されるために、実現したいソフトウェアを制作できる能力をつけることが困難で、他人の作ったプログラムを理解し、それを改編する程度の教育しか行なわれてこなかった。「言霊」の完成と教育法の開発により、プログラミング教育が大きく変わるものと期待される。

日本語で表現することが苦手な情報処理技術者には、「言霊」のような日本語プログラミング言語は使いにくいという評価が与えられがちである。一方、日本語がうまく書けない情報処理技者の存在が問題になっている。プログラムを日本語で書くことで、日本語の苦手な技術者の意識が変わる可能性がある。

プログラムに無縁の文科系研究者には、「言霊」のようなプログラミング言語は歓迎される。このような言語が利用可能になるなら、自分でもプログラムを書いてみたいという希望を述べる人が多い。これに成功すれば、コンピュータと人間の関係を大きく変える可能性がある。

これまで、自然言語によるコンピュータへの指示は、データベースの問合せ言語などで試みられてきたが、広く利用されるま

でには至っていない。「言霊」の利用者が広がることによって、一般人のコンピュータ理解が深まり、情報社会の在り方が変わる可能性もある。

参考文献

[1] 松澤、中鉢、岡田、大岩：オブジェクト指向技術者養成のためのカリキュラム、情報処理学会「コンピュータ教育」研究会 64-1、2002 年

[2]水谷静夫:「朱唇の手引き」東京女子大学 日本文学科 107pp. 1989 年

[3]芹沢浩:「日本語 Logo 入門」森北出版(株)、183pp. 1994 年

[4]Jon Meyer, Troy Downing：「JAVA バーチャルマシン」 オライリー・ジャパン、480pp. 1997 年

[5] 兼宗進、中谷多哉子、御手洗理英、福井眞呉、久野靖：「オブジェクト指向言語「ドリトル」を利用した情報教育について」 情報処理学会「情報教育」シンポジウム Vol.2001,No.9 pp.275-282 2001 年

[6]木村明、片桐明：「日本語プログラミング言語『**Ｍｉｎｄ**』について その概要と、日本語プログラミングの実用性」 プログラミング言語 16-4、情報処理学会 pp.25-32 1988 年

[7]Bertrand Meyer:「プログラミング言語理論への招待」 アスキー、**450pp. 1995** 年