プログラミング言語　7－3
（１９８６．７．１８）

PostScriptの紹介

小方一郎
電子技術総合研究所

PostScript の紹介をします。PostScript 言語はグラフィックスの能力を持ったインタラクティブなプログラミング言語です。これの主な応用は、文章だけではなく、画像イメージや図を含むようなドキュメントを表現することにあります。PostScript は、レーザビームプリンターなどのいわゆるラスターデバイスを対象として設計されています。

このチュートリアルでは、PostScript 言語としての文法とその意味に付いて述べます。これに加えて、グラフィックスの操作、文書印刷、フォントの扱い等のページ記述能力についても述べます。

POSTSCRIPT Tutorial

Ichiro OGATA
Electrotechnical Laboratory
Tsukuba-science city 305 JAPAN

This is tutorial for the POSTSCRIPT language.

The POSTSCRIPT language is an interpretive programming language with graphics capabilities. Its primary application is to describe printed pages that contains text, figures, images and so on. POSTSCRIPT is designed to deal with the general class of raster output devices, which most likely to be a laser beam printer.

In this tutorial, are discussed syntax and semantics of POSTSCRIPT language, and also its several page description capabilities , including graphics operation, text manipulation, and font description.

1。 PostScript とは

PostScriptは図や表を含むようなドキュメントを作り出すための、いわゆるページ記述言語です。PostScript は出力装置としてLaser Beam Printerなどの、Raster deviceを対象として設計されています。これまではLaser Beam Printer へのインターフェースとしては各々の機器に特有のエスケープ・シークエンス等が用いられてきましたが、PostScript はデバイス・インディペンデントなので、これをプリンターへの標準インターフェースとして用いるという意味もあります。PostScript がこれまでのエスケープ・シーケンス等と決定的に違うのはそれがひとつのプログラミング言語となっている点にあります。 プリンター上では PostScript のインタープリタが通信線から入力されるプログラムを解釈、実行するわけです。一般的な使い方としては、計算機によってPostScriptのプログラムを作りだし、これを通信線を介して Laser Beam Printer に流すことにより、出力を得ます。 PostScript には幅広いグラフィクスのプリミティブとフォントを定義できる能力があるので、TeX, Troff, Scribe等の清書システムの出力や、各種の図形処理の出力として利用できます。

2。PostScript 言語

2.1 実行

PostScript は フォースのような スタック・オリエンテッドな言語です。オブジェクトはリテラルとエグゼキュータブルに分けられます。実行中にリテラルに出会うとそれはオペランドスタックにプッシュされます。エグゼキュータブルに出会えばそれはオペレータとして処理され必要ならばオペランドスタックからオペランドを取り出し実行します。実行の結果はやはりオペランドスタックにプッシュされます。
例えば、PostScript の

```
40 60  add 2 div
```

というプログラムは次のように実行されます。40 にであうとこれはリテラルですから 40 がスタックに積まれます。 60 も同様です。ここで add というオペレータに出会うと add はスタック上の 40 と 60 を取り出し、演算を行い結果の 100 をスタックに積みます。次に 2 がスタックに積まれ、 div が実行され結果 50 がスタックに現れます。

2.2 手続きの定義

次に手続きを定義してみます。

```
/average {add 2 div} def
40 60 average
```

最初に現れた /average は オペレータとしては実行されず、その名前がスタックに積まれます。これが / の効果です。 { と } に挟まれた部分はこれも直接は実行されず、このプログラムが一塊としてスタックに積まれます。ここで def オペレータに出会うことで名前 average にプログラムが代入されるわけです。 { と } で囲まれたプログラムをエグゼキュータブルアレイと呼びます。つまり変数にエグゼキュータブルアレイを代入することが手続きの定義となるわけです。PostScript は名前にであうとそれが変数だと思います。変数の値がリテラルならその値をプッシュします。変数の値がエグゼキュータブルアレイならばそれが実行されるます。ここで

```
/forty 40 def
```

として 40 の代わりに forty という変数を使っても、同じ結果が得られます。この場合 forty の値はエグゼキュータブルではないので forty はオペレータとなりません。

2.3 シンタックス

PostScript はプログラミング言語ですから、普通に印字可能な文字だけで表現されます。

・コメント
%から行末までがコメントになります。

```
例    40 60 average   %  take an average...
```

・数

数には符号付き整数　123 -98 43445 0 +17　及び浮動小数点数 -.002 34.5 -3.62 123e10 1E-5 -1. 0.0 等があります。その外に、基数付き整数 8#1777 16#FFFE 2#1000　があります。基数は 2 から 36 までが使えます。

・文字列
文字列は (と) で囲みます。

例　　(This is a string)
　　(The following is an "empty" string)
　　()
　　(It has 0 () length)

文字列のもう一つの表し方として 十六進数を　< と > で囲む方法もあります。

例　　<901fa3>

は三つのキャラクター でそのコードが　90 1f a3 であるような文字列です。文字列はバイナリーデータとして使われることもあり、この記法はそう言うばあいに有効です。

・名前
特殊文字 (%,(,),<,>,[,],{,},/) 以外の文字列で数ではないものが名前となります。例えば　abc Offset $$ 23A 13-456 a.b $MyDict @pattern　等は正しい名前です。名前の前に / を付けたものをリテラル名と呼びます。役割は、前に述べたように、変数と名前そのものを区別するためのクウォートです。これに対し / を付けない名前をエグゼキュータブル名と呼びます。

・配列
配列は１次元のものだけが許されます。配列は [と] で表します。配列の要素は同じデータ型である必要はありません。

例　　[123 /abc (xyz)]

上の例では　整数 123 、リテラル名 /abc 、文字列 (xyz) を要素とするような配列となっています。勿論要素として配列を含むような配列も許されます。

ここで重要なのは [と] は実は文法の要素ではなく、単なるオペレータであることです。 したがって、[と] の間の部分も普通に実行が進んでいきます。[はスタックに mark と呼ばれる特別なオブジェクトをプッシュし] は スタックの上で　mark までのオブジェクトを集めて配列を作るようなオペレータです。
[1 2 add] を実行して得られるものは [3] と同じものです。

・手続き
{ と } で囲むことで、エグゼキュータブルアレイを作ります。配列のシンタックスと良く似ていますが、実はこれは全然別物です。というのは、 { } の中は実行されないからです。例えば
　　{ add 2 div } は
すぐには実行されず、全体が手続きとしてスタックに積まれます。手続きはエグゼキュータブル名を通してアクセスされたときに起動します。手続きは一種の遅延実行メカニズムとも言えます。

2.4 データ型

PostScript でサポートされるデータ型として次のものがあります。

```
integer real boolean array string
name dictionary
operator file mark null save fontID
```

・integer,real,array,string,name　に ついては、前に述べた通りです。
・boolean は true　と false から成ります。これは if、ifelse などのオペレータ共に実行の制御に用いられます。
・operator は PostScript の世界での”機械語コード”です。これを実行することで組み込み機能が起動されます。
・mark は配列を作り出すときに用いられる特殊なオブジェクトです。null は　未定義状態を表すオブジェクトです。
・save は　PostScript のメモリの状態のスナップショットを表しています。オペレータ save と　restror　共に用い、PostScript の状態を任意の状態

に復帰させます。
・file は、PostScriptが外部にファイルを持つ場合、これと通信するためのオブジェクトです。
・fontID は PostScript がフォントを扱うための特別なデータ構造です。
・dictionary は次に述べます。

2.5 辞書

辞書(dictinary)は PostScript では重要な役割を持っています。辞書は key とvalue の組が詰まっているものです。 key となるオブジェクトには、特に制限はありませんが、名前を用いるのが普通です。value としては任意のオブジェクトが許されます。辞書は単なる連想記憶を提供するだけではありません。実はオペレータの意味は 辞書の value によって決まるのです。つまり辞書は名前の空間の管理に使われているのです。 PostScript は辞書スタックというスタックを持っています。変数の値（オペレータの意味ともいえますが）はその名前をこの辞書スタック上の辞書を上から順番に検索することにより、決定されます。名前の空間を切り替えたいときには辞書を作り、辞書スタックにプッシュします。その新しい辞書に def することで、古い定義は隠されるわけです。辞書スタックの一番底には、systemdict と呼ばれる辞書があります。これは、PostScript に組み込みのオペレータの定義が納められています。その上に有るのが userdict です。 userdict にはユーザーの定義が書き込まれていきます。先ほどの例では、 def オペレータによって userdict に定義が書き込まれたわけです。この二つの辞書は、決してスタック上から取り去ることはできません。つまり PostScript では add のような基本的なオペレータにも何も特別なことはありません。 add は胆に systemdict の中で定義された名前に過ぎません（値は operator です）。ユーザーは新たに add に定義を与えることで、add の意味を変更することが出来ます。
辞書を辞書スタックに積むオペレータは begin 、ポップするのは end です。新しい名前空間が欲しい場合には、

```
/newdict 10 dict def     % 新しい辞書をつくる
newdict begin            % 辞書を辞書スタックにプッシュする
...                      % 新しい環境で実行する
end                      % 辞書をポップする
```

とすればよいわけです。 ... の中で実行された def などは外に持ち出されません。

2.6 制御構造

PostScript では手続きをオペランドとして渡すことで制御構造を実現します。例えば、

```
/a 3 def
/b 5 def
a b gt {a} {b} ifelse
```

を考えてみます。3 、5 がそれぞれプッシュされgtが実行されます。a > b なので、falseがプッシュされます。（3 と 5 は gt によってポップされています）次に手続きが二つプッシュされ ifelse が実行されます。ifelse はスタックからブーレアンと手続き二つをを取り出し、ブーレアンが false なので {b}が実行され５がスタックに積まれます。
その他には、if for repeat loop exit forall pathforall 等のオペレータが定義されています。

2.7 いろいろなオペレータ

スタックを操作するオペレータには次のものがあります。

```
dup exch pop copy roll index
```

算術オペレータには次のものがあります。

```
add sub mul div idiv mod
abs neg ceiling floor round truncate
sqrt exp ln log sin cos atan
rand srand rrand
```

配列、辞書、文字列オペレータには次のものがあります。この三つのオブジェクトは同じメソッドでアクセス出来ます。

get put copy length forall

2.8 大域脱出

大域脱出のために、次のオペレータがあります。

```
{ ... } stopped
{handleerror} {terminate} ifelse
```

{...} の中で stop が実行されるとスタックにtrue が 積まれ直ちにこのレベルに脱出します。そうでないときには false がプッシュされます。 if else により handleerror か terminate が実行されます。

3。グラフィックス

3.1 座標系

PostScript では、単位系として big point (1/72 inch) を用いています。
これを基本として、scale rotate translation の各オペレータが定義されます。

・72 72 scale は、単位系を 72 倍即ち 1 inch に変更します。

・2 2 translate は、原点を 2 inch, 2 inch 斜め、つまり (2,2) の位置に変更します。

・45 rotate は、時計方向に 45 度の回転をおこないます。角度は degree で計ります。

3.2 パス

PostScript では、パスと呼ばれるものがグラフィックスの基本となります。 cp は現在点 (current point) の意味です。

```
/box { newpath      %  パスを初期化する。
       0 0 moveto   %  0 0 に cp を移動する。
       0 1 lineto   %  0 1 に cp を移動し、
                       直線を加える。
       1 1 lineto
       1 0 lineto
       closepath    %  パスを閉じる。
       } def
gsave               %  現在の座標系の保存。
72 72 scale         %  単位系を 1 inch とする。
box fill            %  箱を描き、中を黒く塗る。
grestore            %  座標系を以前の状態に復帰。
```

パスを作り出すオペレータとしては、 lineto の他に arcto, curveto 等があります。パスへのオペレータとしては fill の他に stroke, clip があります。stroke はパスに沿ってある太さで線を描きます。 clip は、パスの内部をクリップエリアとして確立します。

3.3 イメージ

PostScript には、画像データを読み込む能力もあります。イメージデータは、最大 256 段階までのグレイスケールも取り扱えます。画像データそのものは、文字列で表しますが、一般に非常に大きいものになるため、これをファイルから読み込むことがあります。

4。フォント

4.1 テキストの印字

PostScript は、フォントの定義にも特徴があります。フォントは基本的にはそれを作り出すグラフィックオペレーションで定義されています。パスを基本としたものでも、イメージを基本としたものでも構いません。フォントは任意のポイントサイズで使うことが出来ます。

```
/Helvetica findfont 10 scalefont setfont
45 292 moveto
(This is Helvetica font in 10 point) show
```

この例は、組み込みであるヘルベチカというフォントを用いた例です。 10 ポイントを指定しています。（組み込みのフォントはすべて1ポイントで設計されています）文字列で表されたテキストが印字されます。

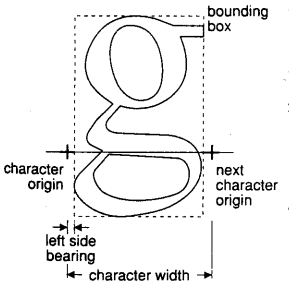
4.2 フォントとパス

フォントはパスで定義されている場合、そのパスを取り出すことも出来ます。例えば、


```
newpath
60 280 moveto
0.5 setgray
(ABC) false charpath
2 setlinewidth
stroke
```

というようにフォントのアウトラインを打ち出すことも出来ます。

4.3 フォント・メトリック



フォントは、一般的なフォント情報とコンパティブルとなっています。これは、TeX で用いているフォント情報もこれと同じものです。フォントは、原点、フォントを取り囲む最小の箱であるバウンディング・ボックス、幅、ベアリング等からできています。このフォント情報は、清書システムがラインブレーキング等を決めるため情報として別のファイルで提供されています。

4.4 フォントのユーザ定義

フォントをユーザが定義するのはやや面倒ですが、勿論可能です。基本的には、フォント辞書を作りだしその中に /BuildChar という特別な手続きを定義すれば良いのです。 /BuildChar の中で、各文字ごとに幅、原点から相対距離で表したバウンディングボックスを決め、パスもしくはイメージで文字の形を描く手続きを実行すれば良いのです。

5。最後に

PostScript は、全体的に良く設計されている言語であると思います。デバイスに独立でもあるので、これが普及すれば、これまで清書システムのネックとなっていたプリンタードライバーの開発が不用となるわけで、 TeX などの普及にも弾みがつくものと思われます。

References

[1] Adobe Systems Incorporated,PostScript language Reference Manual, Addison-Wesley,1985

[2] Adobe Systems Incorporated,PostScript language Tutorial and Cookbook, Addison-Wesley,1985