LATEX タイプセッティングシステムでのパーサ構築の試み

廣　島　　勉

1　ドメイン特化言語

LATEXタイプセッティングシステムにおいて、tabular環境とarray環境のフォーマット指定や、tikz描画パッケージは、埋め込みドメイン特化言語であり、使用中の言語TEXを使って、より狭い特定の領域の問題を、よりエレガントに記述している。

tabular環境とarray環境はフォーマット指定

```
|*{6}{c|}
```

を、TEXの\halignコマンドのプリアンブルへと変換し、tikzパッケージはコマンド

```
\path [draw] (0,0) -- (0,1) -- (1,1) -- (1,0) -- cycle;
```

をPDFの描画命令へと変換する。

ドメイン特化言語はその母体となる言語の字句構造や構文に由来する制約をうけるのが常である。純粋関数型言語Haskell[4][5]や、オブジェクト指向言語Ruby[6]を基盤とする場合、ドメイン特化言語はHaskellやRubyの字句構造、構文構造をおおよそ受け継ぐことになる。

言語を自らの言語で拡張することは、マクロ・システムを持つプログラム言語Lispの得意とするところであった。プログラム言語Common Lispでは、マクロ・システムを用いて、その字句構造と基本的構文構造の上に、新たな構文を定義することができる[1][2][3]。

TEXタイプセッティングシステムのプログラミング言語的要素も､マクロ･システムであり、古くから多くのパッケージャがドメイン特化言語を設計してきた。さらにTEXは他の言語と違い、字句構造すらも自らの操作対象にできる。つまり、TEXでは、ソースコードに埋め込まれた文字列を、字句解析と構文解析の対象にできる。

プログラマがプログラムを容易に書けるようにするプログラムをメタプログラムと言う[2][6]。本論文では、LATEXタイプセッティングシステムにおいて、簡単な字句解析と構文解析を行うマクロ群を構築するのに有用なマクロ群を構築する。

2　パーサーの基本構造

TEXにおいて、

```
\def\foo#1{<replacement string>}
```

のように定義された、パラメータを1つとるコントロールシークエンス\fooが、TEXのソー

スコード中に置かれると、それに続くトークンをパラメータ#1として捕捉し、置換文字列〈*replacement string*〉にマクロ展開される。

パラメータを1つとるマクロを末尾再帰で連続して展開させるのが、パーサ（解析器）の基本構造である。

```
\def\parse#1{%
  \if#1;%
  \else
  \message{(#1)}%
  \expandafter
  \parse
  \fi
}

\parse 1 2 3 4 5 6 ;
```

ここでは、セミコロン;を解析終了のマークとしている。処理\message{(#1)}により端末に

```
(1) (2) (3) (4) (5) (6)
```

が出力される。

パラメータ#1に捕捉されたトークンにしたがってその処理を変化させるのが、パーサーの基本構造となる。

```
\def\parse#1{%
  \if#1;%
  \else
  \ifodd#1\relax
  \message{(#1)}%
  \else
  \message{[#1]}%
  \fi
  \expandafter
  \parse
  \fi
}

\parse 1 2 3 4 5 6 ;
```

```
(1) [2] (3) [4] (5) [6]
```

しかし、このように、TEXのプリミティブ\if、\ifxや\ifodd等を用いて、パラメータ#1に捕捉したトークンを判定するのは、何よりエレガントではなく、解析対象のトークン列にマクロ展開可能なコントロールシークエンスが含まれている場合に問題となる。例えば、

```
\def\X{abc}
```

```
\def\Y{abc}
```

と定義されている場合は、\ifx \X \Yによる比較では、マクロとしての同一性が検査され、\Xと\Yは等価なものと判定されるが、字句解析を行うパーサは、異なるものと判定すべきである。

そこで、TEXの\csname \endcsnameプリミティブを用いて、解析処理を個別に行う必要のあるトークンに対応したディスパッチテーブルを構築する。

```
\def\dispatch#1#2#3{%
  \@ifundefined{#1\string#3}{%
    #2%
  }{%
    \@nameuse{#1\string#3}%
  }%
}

\@namedef{dispatchtable+}{\message{plus}}
\@namedef{dispatchtable-}{\message{minus}}

\dispatch{dispatchtable}{\message{undefined}} +
\dispatch{dispatchtable}{\message{undefined}} -
\dispatch{dispatchtable}{\message{undefined}} *
```

```
plus minus undefined
```

ディスパッチテーブルには、LATEXのマクロ\@namedefを使って、

\csname<*base name*><*token*> \endcsname

となるコントロールシークエンスにトークン〈*token*〉に対応する処理を登録する。

\dispatchはパラメータ#1にディスパッチテーブルの語幹〈*base name*〉を、パラメータ#2にディスパッチテーブルに処理が登録されていない場合のデフォルト処理を指定する。\dispatchはパラメータ#3に解析対象となるトークンを捕捉し、ディスパッチテーブルに登録された処理を実行する。\dispatchを使ったパーサは次になる。コントロールシークエンス\breakを解析終了マークとするため、対応する処理の最後に末尾再帰のための\parseを置いていない。

```
\@namedef{dispatchtable+}{\message{plus}\parse}
\@namedef{dispatchtable-}{\message{minus}\parse}
\@namedef{dispatchtable\string\break}{\message{stop}}
\def\parse{%
  \dispatch{dispatchtable}{\message{undefined}\parse}%
}

\parse + * - \break
```

```
plus undefined minus stop
```

\dispatchの問題点は、パラメータ#3には空白トークンや、TEXのグルーピングトークン{ }を捕捉できないことである。そこで、TEXの\furtureletプリミティブを用いて、空白トークンとグルーピングトークンをフィルターする。

```
\long\def\let@next@token#1{%
  \long\def\reserved@a{#1}%
  \futurelet\@let@token\reserved@a
}

\long\def\gobble@next@space#1 {#1}

\def\parse{%
  \let@next@token{%
    \@firstofone{\ifx\@let@token}
    \expandafter\@firstoftwo
    \else
    \expandafter\@secondoftwo
    \fi
    {% next is space
      \message{space}\gobble@next@space\parse
    }{%
      \ifx\@let@token{\let\@let@token}%
      \expandafter\@firstoftwo
      \else
      \expandafter\@secondoftwo
      \fi
      {% next is grouped
        \message{grouped}\@firstoftwo\parse
      }{%
        \dispatch{dispatchtable}{\message{undefined}\parse}%
      }%
    }%
  }%
}

\parse + * - {\error} \break
```

```
plus space undefined space minus space grouped space stop
```

グルーピングトークン{ }は解析対象のトークンとせず、グループに入れられたトークンは「引用符」に入れられた状態とみなし、パーサの解析対象から外している。

3　パーサ構築マクロ

より本格的なパーサは状態マシンとして構築する。TEXでは解析状態も解析状態遷移も、マクロとして実装する必要がある。解析状態マクロが捕捉したトークンに従って、対応する解析状態遷移マクロを展開する。

```
\def\@define@parse@state@#1#2#3#4#5\space#6\group#7\other#8{%
  \def\reserved@c##1##2##3##4##5##6##7##8##9{{#4}}%
  \def\reserved@b##1##2##3##4##5{%
    #1\gdef#2##2{%
      \begingroup
      \let\bgroup=\@undefined
      \let@next@token{%
        \@firstofone{\ifx\@let@token} \expandafter\@firstoftwo
        \else\expandafter\@secondoftwo\fi
        {% next is space
          \endgroup
          \gobble@next@space{##3}%
        }{%
          \ifx\@let@token{\let\@let@token}\expandafter\@firstoftwo
          \else\expandafter\@secondoftwo\fi
          {% next is grouped
            \endgroup
            ##4%
          }{%
            \endgroup
            \parse@token{#3}{##1}{##5}%
          }%
        }%
      }%
    }%
  }%
  \expandafter\reserved@b\reserved@c
  {{##1}}{{##2}}{{##3}}{{##4}}{{##5}}{{##6}}{{##7}}{{##8}}{{##9}}%
  {#4}{#6}{#7}{#8}%
  \def\reserved@a##1{%
    \def\def@parse@action####1####2####{%
      \expandafter\gdef\csname#3\string####1\endcsname##1####2%
    }%
  }%
```

```
    \expandafter\reserved@a\reserved@c
    {####1}{####2}{####3}{####4}{####5}{####6}{####7}{####8}{####9}%
    #5%
  }
```

マクロ\@define@parse@state@の置換テキスト内の、\gdef#2##2でパーサの解析状態を表すマクロを定義する。ここで定義される解析状態マクロは前節で説明した基本構造をもつ。パラメータ#2は解析状態マクロに結びつけるコントロールシークエンスに、パラメータ##2は、パラメータ#4に由来する、付随する解析情報を保持するための引数リストに当たる。

続いて、マクロ\@define@parse@state@の置換テキスト内で、解析状態遷移を表すマクロを定義するためのマクロ\def@parse@actionを定義する。\def@parse@actionを用いて解析状態遷移マクロを定義するコードは、置換テキストの最後、パラメータ#5の位置に展開される。

解析状態遷移マクロに結びつけるコントロールシークエンスは、解析状態マクロのコントロールシークエンスと解析対象のトークンから合成され、解析状態に付随したディスパッチテーブルを構成する。解析状態遷移マクロは前節の\dispatchにあたるマクロ\parse@tokenを通じて展開される。

```
  \long\def\parse@token#1#2#3#4{%
    \@ifundefined{#1\string#4}{%
      #3#4%
    }{%
      \@nameuse{#1\string#4}#2%
    }%
  }
```

解析状態マクロに付随する解析情報は、マクロ\parse@tokenのパラメータ引数#2を通じて、解析状態遷移マクロに自動的に追加される。マクロ\def@parse@actionにおいても、解析状態遷移マクロの定義に、解析状態マクロを定義する際の解析情報引数リストを追加する。

```
  \def\def@parse@state#1#2#{%
    \@define@parse@state\relax#1{#2}%
  }
  \def\long@def@parse@state#1#2#{%
    \@define@parse@state\long#1{#2}%
  }
  \def\@define@parse@state#1#2#3{%
    \def\reserved@b{\@define@parse@state@#1#2}%
    \edef\reserved@a{{\expandafter\@gobble\string#2}}%
    \expandafter\reserved@b\reserved@a{#3}%
  }
```

マクロ\def@parse@stateと、そのヴァリエーション\long@def@parse@stateが、パーサの解

析状態を表すマクロ定義のフロントエンドとなる。書式は次の通りである。

```
\def@parse@state<control sequence><parameters to hold status> {%
  \def@parse@action<token><additional parameters> {<action for the token>}
  <...>
}\space{%
  <action for the space>
}\group{%
  <action for grouped tokens>
}\other{%
  <action for the others>
}
```

4 RPN計算機

最後に、応用として、逆ポーランド記法の計算機マクロを定義する。0以上の整数を数値リテラルとし、+、-、*、/を整数値の四則演算、~を正負反転の演算子とした。#でスタックの最上段の数値を表示して終了する。

```
\def@parse@state\RPN@#1{%
  \def@parse@action#{\message{\@firstoftwo#1}}%
  \def@parse@action+{%
    \count@\expandafter\@firstoftwo\@secondoftwo#1\relax%
    \advance\count@ by \@firstoftwo#1\relax
    \edef\reserved@a{\noexpand\RPN@{{\the\count@}{\expandafter\@secondoftwo
    \@secondoftwo#1}}}%
    \reserved@a
}%
\def@parse@action-{%
  \count@\expandafter\@firstoftwo\@secondoftwo#1\relax%
  \advance\count@ by -\@firstoftwo#1\relax
  \edef\reserved@a{\noexpand\RPN@{{\the\count@}{\expandafter\@secondoftwo
  \@secondoftwo#1}}}%
  \reserved@a
}%
\def@parse@action*{%
  \count@\expandafter\@firstoftwo\@secondoftwo#1\relax%
  \multiply\count@ by \@firstoftwo#1\relax
  \edef\reserved@a{\noexpand\RPN@{{\the\count@}{\expandafter\@secondoftwo
  \@secondoftwo#1}}}%
  \reserved@a
```

```
    }%
    \def@parse@action/{%
      \count@\expandafter\@firstoftwo\@secondoftwo#1\relax%
      \divide\count@ by \@firstoftwo#1\relax
      \edef\reserved@a{\noexpand\RPN@{{\the\count@}{\expandafter\@secondoftwo
      \@secondoftwo#1}}}%
      \reserved@a
    }%
    \def@parse@action~{%
      \count@\@firstoftwo#1\relax
      \multiply\count@ by -1\relax
      \edef\reserved@a{\noexpand\RPN@{{\the\count@}{\@secondoftwo#1}}}%
      \reserved@a
    }%
    \def@parse@action0{\RPN@digit{#1}{0}}%
    \def@parse@action1{\RPN@digit{#1}{1}}%
    \def@parse@action2{\RPN@digit{#1}{2}}%
    \def@parse@action3{\RPN@digit{#1}{3}}%
    \def@parse@action4{\RPN@digit{#1}{4}}%
    \def@parse@action5{\RPN@digit{#1}{5}}%
    \def@parse@action6{\RPN@digit{#1}{6}}%
    \def@parse@action7{\RPN@digit{#1}{7}}%
    \def@parse@action8{\RPN@digit{#1}{8}}%
    \def@parse@action9{\RPN@digit{#1}{9}}%
  }\space{%
    \RPN@{#1}%
  }\group{%
    \RPN@error
  }\other{%
    \RPN@error
  }%
  \def@parse@state\RPN@digit#1#2{%
    \def@parse@action#{\message{#2}}%
    \def@parse@action+{%
      \count@\@firstoftwo#1\relax
      \advance\count@ by #2\relax
      \edef\reserved@a{\noexpand\RPN@{{\the\count@}{\@secondoftwo#1}}}%
      \reserved@a
    }%
    \def@parse@action-{%
```

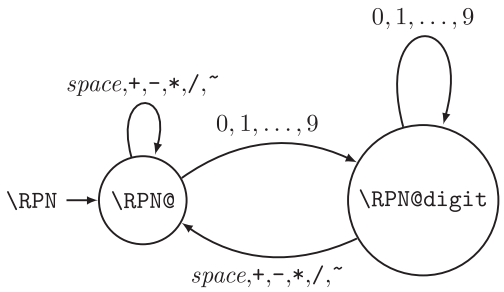
```
      \count@\@firstoftwo#1\relax
      \advance\count@ by -#2\relax
      \edef\reserved@a{\noexpand\RPN@{{\the\count@}{\@secondoftwo#1}}}%
      \reserved@a
    }%
    \def@parse@action*{%
      \count@\@firstoftwo#1\relax
      \multiply\count@ by #2\relax
      \edef\reserved@a{\noexpand\RPN@{{\the\count@}{\@secondoftwo#1}}}%
      \reserved@a
    }%
    \def@parse@action/{%
      \count@\@firstoftwo#1\relax
      \divide\count@ by #2\relax
      \edef\reserved@a{\noexpand\RPN@{{\the\count@}{\@secondoftwo#1}}}%
      \reserved@a
    }%
    \def@parse@action~{%
      \count@#2\relax
      \multiply\count@ by -1\relax
      \edef\reserved@a{\noexpand\RPN@{{\the\count@}{#1}}}%
      \reserved@a
    }%
    \def@parse@action0{\RPN@digit{#1}{#20}}%
    \def@parse@action1{\RPN@digit{#1}{#21}}%
    \def@parse@action2{\RPN@digit{#1}{#22}}%
    \def@parse@action3{\RPN@digit{#1}{#23}}%
    \def@parse@action4{\RPN@digit{#1}{#24}}%
    \def@parse@action5{\RPN@digit{#1}{#25}}%
    \def@parse@action6{\RPN@digit{#1}{#26}}%
    \def@parse@action7{\RPN@digit{#1}{#27}}%
    \def@parse@action8{\RPN@digit{#1}{#28}}%
    \def@parse@action9{\RPN@digit{#1}{#29}}%
  }\space{%
    \RPN@{{#2}{#1}}%
  }\group{%
    \RPN@error
  }\other{%
    \RPN@error
  }%
```

```
\newcommand*\RPN{\RPN@{}}

\RPN 1~ 23 4*+5/#
```

18

マクロ\RPN@と\RPN@digitは次の図に示す状態マシンを構成している。マクロ\RPN@と\RPN@digitの第1パラメータとして、RPN計算機のスタックを保持する仕組みである。\RPN@digitは、数値リテラルを解析中の状態に対応する。



参考文献

［1］Paul Graham（野田　開　訳）、*On Lisp*、オーム社、2005
［2］Doug Hoyte（株式会社タインムインターメディアHOPプロジェクト　訳）、*LET OVER LAMBDA*、株式会社エスアイビー・アクセス、2009
［3］Peter Seibel（佐野　匡俊、水丸　淳、園城　雅之、金子　祐介　共訳）、**実践***Common Lisp*、オーム社、2008
［4］Bryan O'Sullivan, John Goerzen & Don（山下　伸夫、伊藤　勝利、株式会社タイムインターメディア共訳）、*Real World Haskell*—**実戦で学ぶ関数型言語プログラミング**、株式会社オライリー・ジャパン、2009
［5］Jeremy Gibbsons & Oege de Moor編（山下　伸夫　訳）、**関数プログラミングの楽しみ**、オーム社、2010
［6］Paolo Perrotta（角　征典　訳）、**メタプログラミング**Ruby、株式会社アスキー・メディアワークス、2010