Lexer

学生番号 09B20083 吉崎　響

2022 年 10 月 21 日

1　外部仕様

文字列に関して”’”が入力されたにもかかわらず、文字列の終わりを表す”’”が入力される前に改行文字が来てしまった場合、その時点で run 関数を終了する。ts ファイルには”’”が入力されるより前のトークンのみ書き込まれる。
同様に文字列以外でトークンとして認識できないもの (?や!など) が来た場合 run 関数を終了する。ts ファイルには”’”が入力されるより前のトークンのみ書き込まれる。

2　設計方針

pas ファイルの内容を 1 行ずつ解析する。解析には主にプッシュダウン・オートマトンを利用した。使用したオートマトンは図のものである。このオートマトンは 1 トークンを切り出し分類する。つまり 1 トークンを切り出した後は切り出した直後の文字を先頭文字として状態 q0 からスタートする方式になっている。また最初の入力は 1 文字である。一部簡略化のため大雑把に書いている部分がある (スタック=予約語など) が細分化することで正式なオートマトンを構成できるため問題ないと判断している。また空白、タブに関しては、削除するためオートマトンには含まれない。ただし、注釈中に改行が来た場合、以降”}”が入力されるまでオートマトンを利用した解析は行わず読み飛ばす。

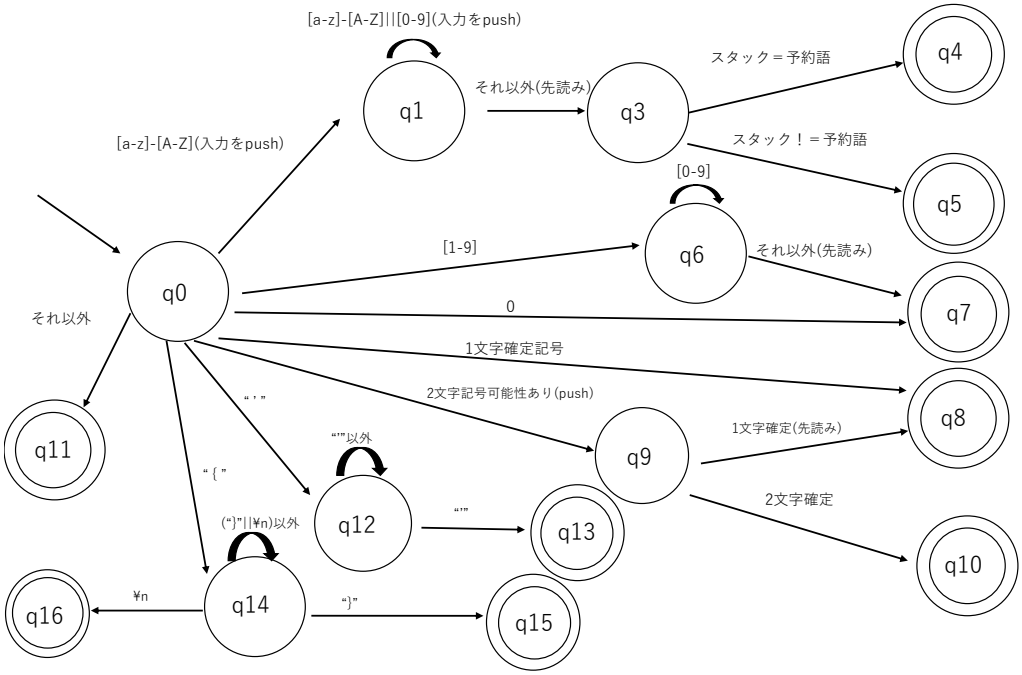

図 1: 使用したオートマトン

2.1 受理状態の説明

q4 が予約語、q5 が識別子、q7 が符号なし整数、q8 が 1 文字の記号、q10 が 2 文字の記号、q13 が文字入力、q15 が注釈、q16 は注釈内での改行、q11 がその他となっている。

2.2 状態遷移の流れ

初期は状態 q0 である。1 文字目がアルファベットであった場合、トークン一覧から分解される可能性があるトークンは、識別子、予約語 (and など) のいずれかである。(文字列は記号 ” ”と”{”関して特別に処理しているため考慮していない) 状態 q1 では、識別子の場合は、トークンの切れ目がアルファベットもしくは数字以外であり、予約語の場合はアルファベット以外であるため、q1 ではアルファベットと数字が入力の間はスタックに入力を push、それ以外の入力の場合は q3 へ遷移となる。q3 ではトークンがスタックに入っている状態である。スタックの中身が予約語と一致する場合 q4 へ、一致しない場合、識別子と認識して q5 となる。
1 文字目が数字であった場合は最初の数字が 0 であった場合、単体でトークンと確定するので q7 へ、1-9 であった場合は q6 へ遷移する。q6 ではトークンの切れ目は数字以外なので、数字以外の入力で q7 へ遷移する。
1 文字目が記号であった場合、その記号が必ず 1 文字で成立する記号かどうかを判断する。1 文字で成立する記号とは=+,-,/,”,(,),[,],;” , ”である。1 文字で成立する場合、q8 へそれ以外の記号の場合は q9 へ移動する。q9 では 1 文字先読みを行い、それが 2 文字記号を成立させる記号であれば q10 へそれ以外であれば q8 へ遷移する。
1 文字目が” ’ ”であった場合、q12 に遷移し、トークンは’ 文字列’ となるため、文字列の終わりを示す” ’ ”が入力されると q13 へ遷移する。1 文字目が” {”であった場合、q14 へ遷移する。} 入力されると q15 へ￥ｎが入力されると q16 へ遷移する。

2.3 受理状態の処理

q4 の場合はどの予約語と一致するかを確認し一致する予約語の ID とトークン名、段落番号を出力する。q5,q7 の場合は切り出したものトークンの種類が確定しているのでそれを出力する。q8,q10 の場合はどの記号と一致するか確認し一致する記号の ID とトークン名、段落番号を出力する。q13 は” ’ ”を含めた文字列と ID、トークン名、段落番号を出力する。q15 は注釈であるため、切り出した注釈部は ts ファイルには反映せず読み飛ばす。q16 は注釈中での改行で”}”が現れるまで解析を行わない。つまり”}”が来るまでファイルの中身を読み飛ばす。(オートマトンを使用中止)
q11 はエラーである。(トークンでも注釈でもないため識別不可) この場合 pas ファイルの解析をやめ現時点での ts ファイルが完成品となる。

3 実装プログラム

3.1 関数説明

int analysis(String int BufferWrier)
与えられた行をトークンに分解し、それぞれの分類関数に渡す。String にはファイル 1 行分の文字列、int には段落番号、BufferWrier は ts ファイルのものである。また retrun が 0 であれば正常終了、1 であれば注釈中の改行、2 であればエラーによる解析中止を表す。alphabet_recoder(String int BufferWrier)
String には識別子または予約語の文字列が格納されており、格納されたものがどのトークン当たるか分析し、その情報 (ID など) を recoder() へ渡す

number_recoder(String int BufferWrier)
String には整数が入っているので整数のトークン情報を recoder() へ渡す
symbol_recoder(String int BufferWrier)
String には記号が入っており、それを分類して情報を recoder() に渡す
text_recoder(String int BufferWrier)
String には”’ 文字列”’が入っており、文字列のトークン情報を recoder() へ渡す
recoder(String String int int BuffrerWrite)
ts ファイルに情報を書き込む
int serch(String)
”}”を探すための関数。与えられた行に”}”がなかった場合 -1 を返す。”}”を見つけた場合、”}”の 1 つ後ろのインデックス番号を返す。

3.2 プログラムの流れ

プログラムの大まかな流れは、pas ファイルから 1 行ずつ中身を読み込み analysis に渡す。analysis が 0 を返す間はオートマトンを利用した分解を行い、適切な関数へ分トークンを渡す。つまり、alphabet_recoder、number_recoder、symbol_recoder、text_recoder のいずれか適切なものにトークンを渡す。するとそれぞれの関数は与えられたトークンを分類し、ID、トークン名、段落番号を recoder() に渡す。そして、recoder が ts ファイルに書き込みを行う。analysis 関数が 1 を返した場合、つまり注釈中の改行が来た場合、”}”が来るまで analysis 関数は呼ばれなくなる。serch 関数で”}”を判定し、それ以降からの文字列を analysis 関数に与えることで解析を再開させる。analysis が 2 を返した場合 run 関数を終了する。

3.3 analysis 関数

```
analysis{
        while(1行の終わり) {
                if(先頭文字がアルファベット){                                                    状態q1
                        while(次がアルファベットか数字){1文字ずつ解析                    状態q2  }
                        トークン切り出し                                  状態q3
                        alphapet_recoder(トークン) ;
                        }
                else if(先頭が数字){
                        if(先頭が0){   number_recoder(0);                                  状態 q7  }
                        else       {
                                     while(次が数字) {                                    状態 q6  }
                                     number_recoder(獲得した数字);                        状態 q7
                                   }
                else if(記号){
                        if(1文字確定記号){ symbol_recoder(入力記号);状態 q8}
                        else if(“ ‘ ”){
                                  while(“ ‘ “以外){
                                                if(行末尾が来る){ return 2;}    /*文字列の終わりがない*/
                                          }
                                   texte_recoder(入力);                                   状態 q13
                                     }
                        else if(“{”){
                                   while(“}”以外){
                                                 if(行末尾){ retun 1; }/*注釈中に改行*/
                                                }
                             }
                                                 .
                                                 .
                                                 .
                        else if(“>”){
                                   if(次が=){ symbol_recoder(>=);}
                                  }
                    }
                else{
                        if(空白){
                                  while(空白){}  /*空白を読み飛ばす*/
                                }
                        else if(改行文字){
                                                 行の終わり
                                  }
                        else{           return 2;    /*トークンにない エラー*/    }
                    }
        }
```

図 2: プログラム概要ソース

analysis は図のようなソースコードになっている。まず最初の文字に関して if 文がオートマトン q0 からの遷移に該当する。その if 文中でそれぞれをトークンに分割している。詳しくは図に記載している。オートマトンには記載していなかったが、空白を消去する処理と改行文字を読み込む処理もここで行っている。また 1 文字確定記号は、あらかじめ 1 文字でしか成立しない記号のリストを作成し一致するものがあるか確かめることで実現している。

3.4　トークンの分類

alphabet_recoder、number_recoder、symbol_recoder、text_recoder にはそれぞれ決まった種類のトークンが引数として渡されている。このうち number_recder と text_recoder は分類の作業は行っていない。分類方法は引数として受け取った String 型のトークンに対して主に swich を利用して実行している。(オートマトンのスタックと比較する部分に当たる) symbol_recoder はすべての記号に対して一致するものを switch 文で探す。alphabet_recoder はすべての予約語に対して一致するかを switch 文で確かめる。一致するものがなければ識別子として認識する。

3.5　1 文字ずつ読み込むには

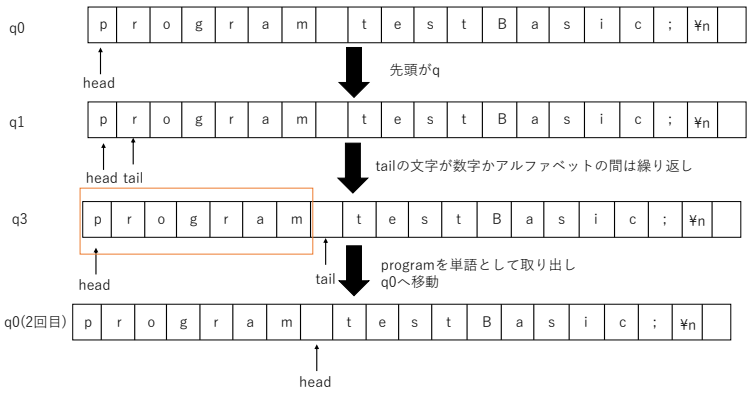

図 3: プログラム概要ソース

図のように文字列としてファイルの 1 行を獲得し、読み始め (トークンの先頭) を head とし、トークンの切れ目まで tail を増やしていく。そして tail が切れ目を指すと分解できるようになり head tail-1 までを 1 つのトークンとして取り出し、次のオートマトンのスタートとして head=tail とし、これを繰り返すことで分解する。

4　考察工夫

今回は字句解析をトークンの分解と分類の大きく 2 つの工程分けようとした。主な理由が 2 つある。まず段階ごとに分けられるためプログラムが設計しやすいと同時にデバックの際切り分けて考えられるという点、もう 1 つは仕様の解釈を誤っていた場合に修正しやすいという点である。実際、識別子はアルファベットのみであると勘違いしており、数字を考慮していない設計になってしまっていたが、トークンの切り出し方のみ修正すればよかったため、修正が容易であった。
ただ 2 段階に分けることには欠点もあり、トークンとして分解するためにはある程度、どの種類トークンであるか区別しておく必要があるため、分類と分解を完全に独立させることができないという点である。今回は全体としての方針を統一させるため、すべてのトークンにおいて分解と分類の 2 段構成で行ったがこ

れは冗長を含む構成になっている。というのは記号を分解する際に、特定の記号に対して、特定の記号が続かなければならない場合、分類と分解が同じになってしまう (: が 1 文字目に来た場合、2 文字目をトークンとして含むかどうかは次を見ないと分からない。ここで次が=の場合に初めて 2 文字で 1 つのトークンとわかるが、この時点で分類も完了してしまっているということ) 今回は 2 文字で切り取って再度分類してしまったため実行効率は 1 段階のほうがよいと考えられる。

5 感想

オートマトンの実用性を痛感できた。例えば、以前であればプログラムを組んでいる時、すべての条件を網羅できているかを主に記憶で確認していくしかなかったが、オートマトンを利用したことで条件の網羅性が認識しやすくなったうえ、レポートで作成したプログラムを説明する際もオートマトンが基盤となっているため、非常に説明がスムーズになった。また Java スクリプト特有の難しさがわかった。C 言語と異なり、java は import することで様々なライブラリ関数を使える。非常に便利な関数も多く使い勝手が良いかった。しかし、便利ゆえに関数の細かい仕様を把握していないがためにデバックに苦労する場面が多かった。例えば、length() はヌル文字を含まない文字列の長さを返すものであるのに、インデックス番号を返すと勘違いしたことによって配列内でずれが生じていたりした。こういったことを防ぐためには使う前にその関数に関する記事を読む必要があると感じた。

6 謝辞

演習 D では help チャンネルでの迅速な対応ありがとうございました。