Euro Tcl 2006

Development and usage of C, Java and Tcl based scanners in Tcl-applications

Dr. Detlef Groth and Alexander Straub, MPIMG Berlin, Germany, dgroth(at)molgen.mpg.de

Abstract: Although since many years many scanner generation frameworks like flex (C), jflex (java) and fickle or ylex (Tcl) are existing most of the scanners and parsers are still handwritten. This is this case although it is well proven that automatically generated scanners are easier to write and to maintain and are at least as fast as handwritten ones. Tcl lexers like fickle and ylex are Tcl-only implementations and are well suited for relative small files. In bioinformatics however we are working often with files several gigabytes in size. Here tcl-based scanners are offering only a poor performance if compared with C- or java based scanners. In order to overcome those limitations we were trying to utilise C-based scanners inside tcl-programs using the critcl package and java based scanners using the jacl/tcljava package with promising results.

The current scanner generator packages for tcl, ylex, fickle and tcLex have some serious drawbacks thereof limiting its usability. tcLex is a old C-extension and not actively maintained. ylex requires that the whole string which will be scanned must be in memory which is not suitable for large files. Both ylex and tcLex use more a tclish than a lex-like syntax to dynamically construct scanners. The scanners can be saved as tcl-code as well to generate scanners which can be used independently from the ylex-package. Fickle in contrast use lex-like input files and generates only static scanners which can be used independently from the fickle command line application. In order to overcome the global type of variables and procedures I recently developed ifickle which generates itcl-classes. Those scanners can be integrated into a larger framework/package like the biotcl-package. However due to its heavy use of the regexp command both ylex and fickle/ifickle are generating very slow scanners if the file size increases. Ifickle based scanners have a slightly larger startup overhead and are slightly less slow on larger files.

Table 1 compares the performance of various wc-scanners either hand crafted or generated with scanner generation frameworks like perl-parseLex, flex or re2c. As it visible performance of ylex and fickle / ifickle based scanners is not acceptable if larger files as often required if analysing biological data like genomes, proteomes.

In order to overcome the slow performance with scanners generated by tcl-based scanners generators, we were investigating the usability of C and Java based scanners for integration into tcl-programs. From the two C-based scanners, flex and re2c we choose re2c for an wc-implementation. The reason was, that in contrast to flex, re2c does not depend from external libraries and the code generated from re2c can thereof be easily integrated into tcl-programs using the critcl-package. Furthermore it seems that from our data and from the data of others re2c based scanners are 2 - 5 times faster than flex based scanners.

Starting to use re2c was somehow difficult, but after fixing some initial issues with buffer filling and wrapping the re2c application into the critcl-application it was possible to use the re2c scanner either standalone or via the tcl application. The performance was comparable with the pure re2c application however the startup cost was with 1 second for small files quite high.

Because coding in C is for many people much more difficult than coding in java we were further investigating the possibility to utilise java based scanners inside tcl. There are two possible choices for communicating between the java machine and the tcl application. Recently there has been some effort to compile the tclblend library with stubs to utilise it inside starkits, however there is currently no starkit enabled tclblend library available. Tclblend would enable tcl programmers to directly communicate with java classes like with tcljava. In our study we were utilising a second approach, communication via sockets. The tcl application is looking for a free socket port and starts the java application. The java application sends it's parsing results via the socket connections to the tcl application. This approach has bigger startup penalty, because two interpreters must be started, but for large files which should be scanned, the startup is less import than the actual scanner performance. As can be seen from table 1 the scanning speed is comparable with a re2c based wc scanner. Both are about 3 orders of magnitude faster in comparison to tcl based scanners.

We were furthermore writing scanners with ifickle, re2c and jflex for biological data (Blast). Table 2 summaries our observations. The speed for tcl based scanners on larger files is very slow, however Coding and Integration with pure tcl based scanners is easier to accomplish in comparison with re2c or java based scanners.

Conclusions

re2c and jflex based scanners can be utilised for tcl-application thereof limiting the bad performance of current tcl based scanner generators. The big speed difference might be due to the fact that both jflex and re2c did not use any regexp library, but rather are building its own effective mechanism to translate regular expressions into a fast scanner. It might be therefore of interest to study the code generated by re2c and especially for jflex and trying to build a similar tcl based scanner generator. Using the critcl/re2c or socket/java approach gives some startup penalties but this is not of importance if large amount of data, as common in biology, needs to be scanned.

References

- tcLex http://membres.lycos.fr/fbonnet/Tcl/tcLex/index.en.htm
- fickle http://tcl.jtang.org/fickle/
- ylex http://www.fpx.de/fp/Software/Yeti/
- re2c http://re2c.sourceforge.net/
- flex http://www.gnu.org/software/flex/
- jflex http://jflex.de/
- tclblend http://tcljava.sourceforge.net/

Tables

Table 1:

Mode/Size(byte)	1	10	100	1000	10000	100000	1000000	1000000
tcl-hand	0.130	0.120	0.200	0.290	0.135	0.489	3.877	nd
tcl-yeti-dynam	0.531	0.561	0.574	0.643	1.538	9.765	91.706	nd
tcl-yeti-static	0.454	0.466	0.474	0.551	1.467	9.491	89.334	nd
tcl-fickle	0.129	0.224	0.133	0.184	0.487	3.460	33.099	nd
tcl-ifickle	0.439	0.435	0.451	0.489	0.713	2.801	23.947	nd
python-hand	0.724	0.258	0.260	0.248	0.260	0.881	1.739	nd
perl-hand	0.013	0.012	0.013	0.012	0.015	0.043	0.321	nd
perl-parseLex	0.213	0.203	0.229	0.258	0.437	2.359	21.998	nd
java-hand	0.770	0.475	0.502	0.499	0.496	6.101	5.748	nd
java-jflex	0.579	0.482	0.567	0.488	0.577	0.661	0.676	1.584
tcl-java-jflex	1.003	1.001	1.002	0.996	0.998	1.014	1.104	1.983
flex-wc	0.006	0.005	0.005	0.006	0.012	0.041	0.376	3.633
wc	0.005	0.080	0.006	0.007	0.009	0.007	0.020	0.148
re2c	0.011	0.009	0.009	0.009	0.011	0.027	0.185	0.618
tcl-critcl-re2c	1.489	1.485	1.316	1.313	1.396	1.388	1.421	1.524

Table 2:

-	Speed	Coding	Integration	Startup
ifickle	--	+++	+++	+
re2c	+++	+	++	(+)
jflex	++	++	+	(+)

Source Code

iwc-fickle.fcl

Implementation of wc with ifickle.

```
%{
#!/usr/bin/tclsh8.4
public variable nline 0
public variable nword 0
public variable nchar 0
%}
%buffersize 1024
%%
\n { incr nline; incr nchar 2 ; }
[^ \t\n]+ { incr nword; incr nchar $yyleng ;}
. { incr nchar;}
%%
if {[llength $argv] == 0} {
   puts stderr "usage wc-fickle inputfile"
   exit 0
}
if {[catch {open [lindex $argv 0] r} yyin]} {
   puts stderr "Could not open [lindex $argv 0]"
   exit 0
}
set sc [iwcfickle \#auto -yyin $yyin]
$sc yylex
puts [format "%7d %7d %7d %s" \
    [$sc cget -nline] \
    [$sc cget -nword] \
    [$sc cget -nchar] [lindex $argv 0]]
close $yyin
```

WC.flex

Implementation of wc with java-jflex'''

```
/* WC.flex */
%%
%public
%class Wc
%standalone
%unicode
%{
 int nchars = 0;
 int nwords = 0;
 int nlines = 0;
%}
%eof{
 System.out.println("   "+nlines+"\t"+nwords+"\t"+nchars);
%eof}
%%

[\n] { nlines += 1; nchars += 1;  }
[ \t]+ { nchars += yylength() ; }
[^ \t\n]+ { nwords += 1; nchars += yylength(); }
```

---

wc-critcl.tcl

WC with re2c, tcl and the critcl package

```
/* File: wc-critcl.tcl */
source ./critcl.kit
package require critcl
set cfile [file rootname [info script]].c
```

```
if [catch {open $cfile r} infh] {
   puts stderr "Cannot open $cfile : $infh"
   exit
} else {
   set ccode [read $infh]
   close $infh
}
critcl::ccode $ccode
critcl::cproc lines {} int { return numline; }
critcl::cproc words {} int { return numword; }
critcl::cproc chars {} int { return numchar; }
critcl::cproc myscan {char* filename} void { readFile(filename); }
if {[llength $argv] != 1 || ![file exists [lindex $argv 0]]} {
   puts "Usage: [info script] "
   exit 0
}
myscan [lindex $argv 0]
puts [format "%7d %7d %7d %s" [lines] [words] [chars] [lindex $argv 0]]
```

---

wc-critcl.re

re2c wc application ready for usage inside tcl applications with the critcl package.

```
/* File: wc-critcl.re */
#include
#include
#include
#include
#define    EOI    319
#define    BSIZE    8192
#define    YYCTYPE        uchar
#define    YYCURSOR    cursor
#define    YYLIMIT        s->lim
#define    YYMARKER    s->ptr
#define    YYFILL(n)    {cursor = fill(s, cursor);}
#define    RET(i)    {s->cur = cursor; return i;}
typedef unsigned int uint;
typedef unsigned char uchar;
int numline, numchar,numword= 0;
typedef struct Scanner {
   int           fd;
   uchar        *bot, *tok, *ptr, *cur, *pos, *lim, *top, *eof;
   uint        line;
} Scanner;

uchar *fill(Scanner *s, uchar *cursor){
   if(!s->eof) {
       uint cnt = s->tok - s->bot;
       if(cnt){
           memcpy(s->bot, s->tok, s->lim - s->tok);
           s->tok = s->bot;
           s->ptr -= cnt;
           cursor -= cnt;
           s->pos -= cnt;
           s->lim -= cnt;
       }
       if((s->top - s->lim) < BSIZE){
           uchar *buf = (uchar*) malloc(((s->lim - s->bot) + BSIZE)*sizeof(uchar));
           memcpy(buf, s->tok, s->lim - s->tok);
           s->tok = buf;
           s->ptr = &buf[s->ptr - s->bot];
           cursor = &buf[cursor - s->bot];
           s->pos = &buf[s->pos - s->bot];
           s->lim = &buf[s->lim - s->bot];
           s->top = &s->lim[BSIZE];
           free(s->bot);
           s->bot = buf;
       }
       if((cnt = read(s->fd, (char*) s->lim, BSIZE)) != BSIZE){
           s->eof = &s->lim[cnt]; *(s->eof)++ = '\n';
       }
```

```
        s->lim += cnt;
    }
    return cursor;
}

int scan(Scanner *s){
    uchar *cursor = s->cur;
std:
    s->tok = cursor;
/*!re2c
    [ \t]+ {
        numchar += YYCURSOR-s->tok;
        goto std;
    }
    "\n" {
        if(cursor == s->eof) RET(EOI);
        s->pos = cursor; s->line++;
        numline++;
        ++numchar;
        goto std;
    }
    [!-~]+ { ++numword; numchar += YYCURSOR-s->tok; goto std; }
    [\000] { RET(EOI) ;}
*/
}
int readFile (char* filename) {
    Scanner in;
    int t;
    memset((char*) &in, 0, sizeof(in));
    in.fd = open(filename, O_RDONLY);
    if (in.fd == -1) {
         printf("error reading file: %s ", filename);
         return 1 ;
    }
    while((t = scan(&in)) != EOI){    }
    close(in.fd);
    return 0 ;
}
int main(int argc, char *argv[]){
    int ret ;
    if(argc != 2) {
        printf("usage: %s ", argv[0]);
        return 0 ;
    }
    ret = readFile(argv[1]);
    printf("\t%i\t%i\t%i\t%s\n", numline,numword,numchar,argv[1]);
    return ret ;
}
```

This re2c code needs to be compiled in the following way:

```
$ re2c wc-critcl.re > wc-critcl.c
# (optional) compile command line application
$ gcc -o wc-critcl wc-critcl.c
# (optional) run command line application
$ ./wc-critcl wc-critcl.tcl
      21      106    694 wc-critcl.tcl
# tcl application
$ tclkit wc-critcl.tcl wc-critcl.tcl
      21     106     694 wc-critcl.tcl
$ wc wc-critcl.tcl
      21     106     694 wc-critcl.tcl
```