Constructing Lexers with Ragel & Objective-C

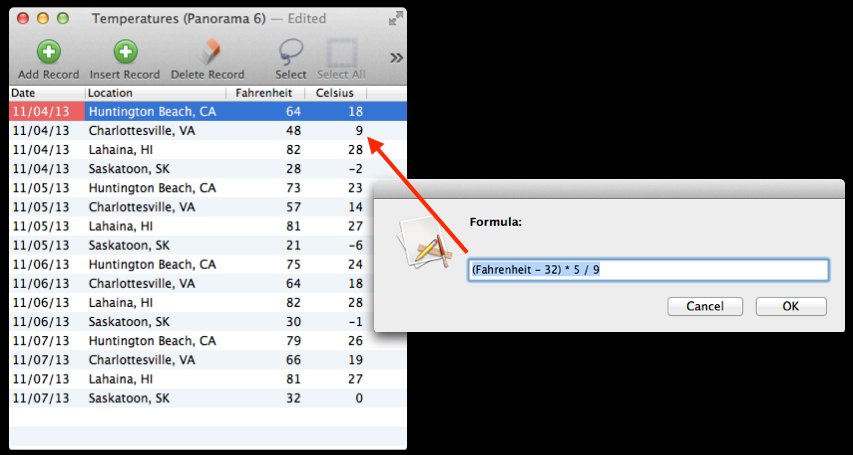
Jim Rea
ProVUE Development

Typical Applications

1) Formula evaluation
   - Spreadsheets
   - Databases
   - Programming languages

2) Other Text Based Protocols
   - CSV
   - HTTP

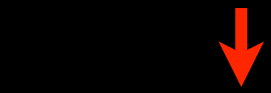
Database Calculations



Lexical Analysis

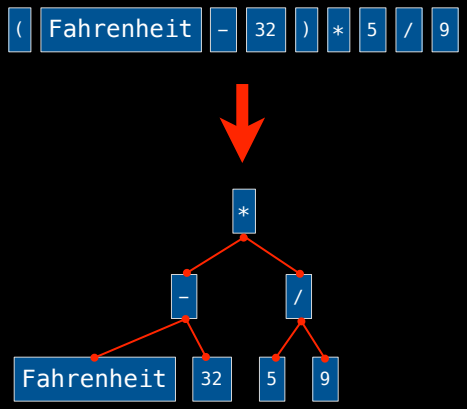
Convert free flow text into a sequence of tokens

```
(Fahrenheit – 32) * 5 / 9
```



Parsing

Converts sequential data into a data structure



Implementing a Tokenizer by Hand

1) Character by character
2) Using NSScanner
3) Using NSRegularExpression

all of these techniques can be
tedious and error prone :(

Automated Tools

1) lex
2) yacc
3) bison
4) flex
5) cup
6) lemon

and dozens more…

http://en.wikipedia.org/wiki/Comparison_of_parser_generators

Ragel

http://www.complang.org/ragel/

- Creates state machines for tokenizing text
  - Targets C, C++, Objective-C, D, Java and Ruby
- Written by Dr. Adrian Thurston
- First released January 2002
  - Latest Update February 2013

Why Ragel?

- Targets C/Objective-C
- Supports 16 bit characters (Unicode)
- No external depencies
  (no external library to link to)
- Flexible licensing
  (no restrictions on generated code)
- Fast runtime execution
- Can be integrated into XCode
- 12 year track record, active community

only package I could find with this combination -- other unicode savvy tools required linking with a GPL'd library

Installing Ragel

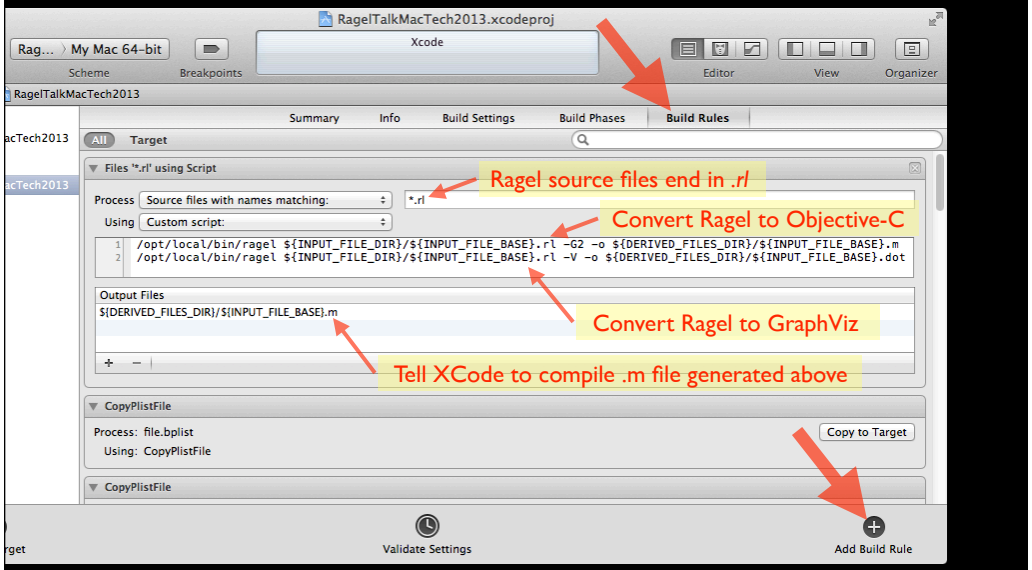
1) Install MacPorts

http://www.macports.org/install.php

2) Install Ragel (using Terminal.app)

```
sudo port install ragel
```

Adding Ragel to your project



More about Xcode Build Rules

http://www.cocoawithlove.com/2010/02/custom-build-rules-generated-tables-and.html

Adding a Ragel Source File

1) File > New > File…

2) Template: Other > Empty

3) Filename ends with .rl

```
test.rl
formulas.rl
parser.rl
```

Empty Ragel State Machine

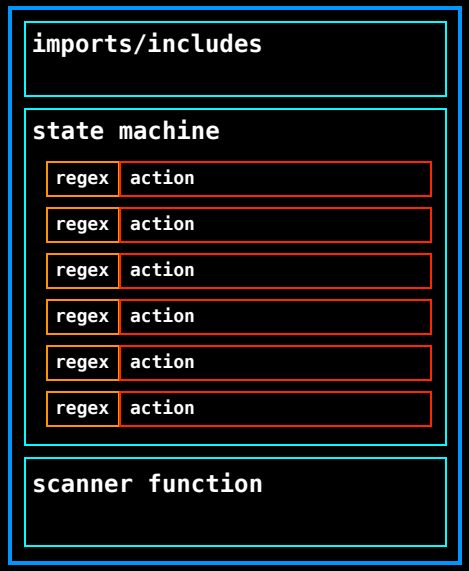
```
#import <Foundation/Foundation.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

%%{

  machine test_lexer;

}%%

%% write data;
```

Everything else is host code (in this case Objective-C)

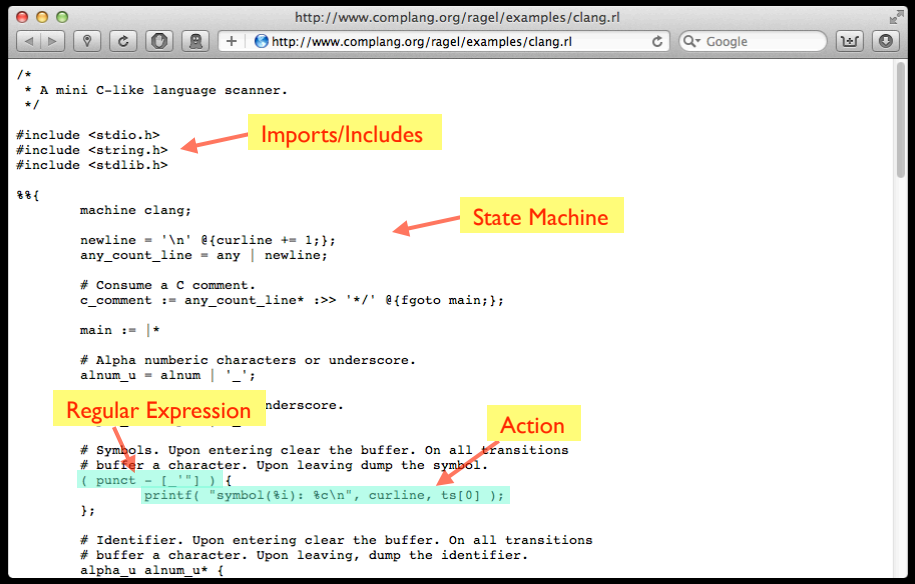
Ragel code is inside %%{ … }%% blocks

Ragel directives on lines beginning with %%

Anatomy of a Ragel Source File

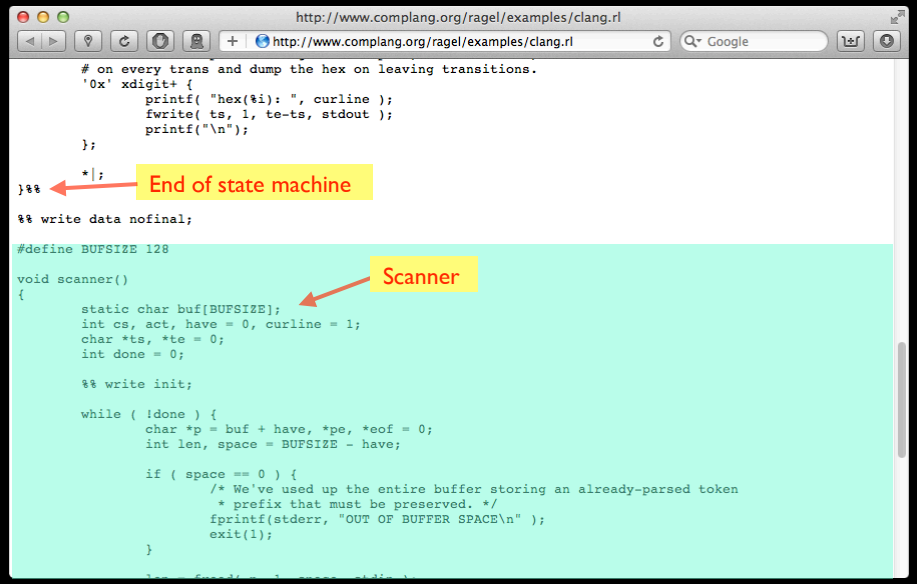


Sample “Mini-C” Language Scanner




```
/*
 * A mini C-like language scanner.
 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

%%{
        machine clang;

        newline = '\n' @{curline += 1;};
        any_count_line = any | newline;

        # Consume a C comment.
        c_comment := any_count_line* :>> '*/' @{fgoto main;};

        main := |*

        # Alpha numberic characters or underscore.
        alnum_u = alnum | '_';

        # Symbols. Upon entering clear the buffer. On all transitions
        # buffer a character. Upon leaving dump the symbol.
        ( punct - [_'"] ) {
                printf( "symbol(%i): %c\n", curline, ts[0] );
        };

        # Identifier. Upon entering clear the buffer. On all transitions
        # buffer a character. Upon leaving, dump the identifier.
        alpha_u alnum_u* {
```

Sample “Mini-C” Language Scanner




```
        # on every trans and dump the hex on leaving transitions.
        '0x' xdigit+ {
                printf( "hex(%i): ", curline );
                fwrite( ts, 1, te-ts, stdout );
                printf("\n");
        };

        *|;
}%%

%% write data nofinal;

#define BUFSIZE 128

void scanner()
{
        static char buf[BUFSIZE];
        int cs, act, have = 0, curline = 1;
        char *ts, *te = 0;
        int done = 0;

        %% write init;

        while ( !done ) {
                char *p = buf + have, *pe, *eof = 0;
                int len, space = BUFSIZE - have;

                if ( space == 0 ) {
                        /* We've used up the entire buffer storing an already-parsed token
                         * prefix that must be preserved. */
                        fprintf(stderr, "OUT OF BUFFER SPACE\n" );
                        exit(1);
                }
```

Simplified scanner() function

```
void scanner()
{
    // internal scanner( variables
    int len, space = BUFSIZE, curline = 1;
    static char buf[BUFSIZE];  // assume big enough for any input

     // Variables used for communicating with Ragel
    char *p = buf;             // pointer to the character data to process
    char *pe;                  // pointer to the end of the data to process
    char *eof;                 // same as pe unless using buffered input
    char *ts, *te = 0;         // start of token, end of token
    // Variables used internally by Ragel
    int cs = 0;                // current state
    int act = 0;               // used internally by Ragel

    %% write init;             // initialize state machine

    len = fread( p, 1, space, stdin);
    pe = p + len;
    eof = pe;

    %% write exec;         // run state machine
}
```

Simplified scanner() function

```
void scanner()
{
    // internal scanner( variables
    int len, space = BUFSIZE, curline = 1;
    static char buf[BUFSIZE];  // assume big enough for any input

    // Variables used for communicating with Ragel
    char *p = buf;             // pointer to the character data to process
    char *pe;                  // pointer to the end of the data to process
    char *eof;                 // same as pe unless using buffered input
    char *t
    // Vari
    int cs
    int act

    %% write init;             // initialize state machine

    len = fread( p, 1, space, stdin);
    pe = p + len;
    eof = pe;

    %% write exec;          // run state machine
}
```

Data Pointer – In C code this variable is expected to be a pointer to the character data to process. It should be initialized to the beginning of the data block on every run of the state machine.

Simplified scanner() function

```
void scanner()
{
    // internal scanner( variables
    int len, space = BUFSIZE, curline = 1;
    static char buf[BUFSIZE];  // assume big enough for any input

    // Variables used for communicating with Ragel
    char *p = buf;              // pointer to the character data to process
    char *pe;                   // pointer to the end of the data to process
    char *eof;                  // same as pe unless using buffered input
    char *ts, *te = 0;          // start of token, end of token
    // Vari
    int cs =
    int act

    %% write init;              // initialize state machine

    len = fread( p, 1, space, stdin);
    pe = p + len;
    eof = pe;

    %% write exec;          // run state machine
}
```

Data End Pointer – This should be initialized to *p* plus the data length on every run of the state machine.

Simplified scanner() function

```
void scanner()
{
    // internal scanner( variables
    int len, space = BUFSIZE, curline = 1;
    static char buf[BUFSIZE];  // assume big enough for any input

    // Variables used for communicating with Ragel
    char *p = buf;              // pointer to the character data to process
    char *pe;                   // pointer to the end of the data to process
    char *eof;                  // same as pe unless using buffered input
    char *ts, *te = 0;          // start of token, end of token
    // Variables used internally by Ragel
    int cs = 0;
    int act;

    %% write init;              // initialize state machine

    len = fread( p, 1, space, stdin);
    pe = p + len;
    eof = pe;

    %% write exec;          // run state machine
}
```

End of File – This should be set to *pe* when the buffer block being processed is the last one, otherwise it should be set to null.

Simplified scanner() function

```
void scanner()
{
    // internal scanner( variables
    int len, space = BUFSIZE, curline = 1;
    static char buf[BUFSIZE];  // assume big enough for any input

    // Variables used for communicating with Ragel
    char *p = buf;              // pointer to the character data to process
    char *pe;                   // pointer to the end of the data to process
    char *eof;                  // same as pe unless using buffered input
    char *ts, *te = 0;          // start of token, end of token
    // Variables used internally by Ragel
    int cs = 0;                 // current state
    int act

    %% writ

    len = fread( p, 1, space, stdin);
    pe = p + len;
    eof = pe;

    %% write exec;          // run state machine
}
```

Token Start, Token End – These are set by Ragel as the input is analyzed, they will be used by our custom state machine actions.

Simplified scanner() function

```
void scanner()
{
    // internal scanner( variables
    int len, space = BUFSIZE, curline = 1;
    static char buf[BUFSIZE];  // assume big enough for any input

    // Variables used for communicating with Ragel
    char *p                                                    process
    char *p                                                  o process
    char *e                                                    input
    char *t
    // Vari
    int cs = 0;                 // current state
    int act = 0;                // used internally by Ragel

    %% write init;              // initialize state machine

    len = fread( p, 1, space, stdin);
    pe = p + len;
    eof = pe;

    %% write exec;          // run state machine
}
```

The *write init* directive causes Ragel to emit initialization code. This should be executed once before the machine is started. At a very minimum this sets the current state to the start state. If other variables are needed by the generated code, they are also initialized here.

Simplified scanner() function

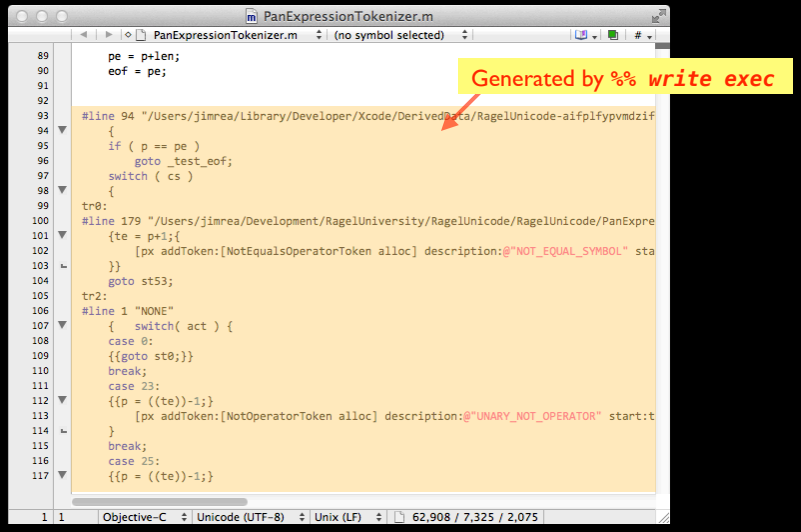
```
void scanner()
{
    // internal scanner( variables
    int len, space = BUFSIZE, curline = 1;
    static char buf[BUFSIZE];  // assume big enough for any input

     // Variables used for communicating with Ragel
    char *p = buf;             // pointer to the character data to process
    char *pe;                  // pointer to the end of the data to process
    char *eof;                 // same as pe unless using buffered input
    char *ts, *te = 0;         // start of token, end of token
    // Variables used internally by Ragel
    int cs = 0;                // current state
    int act = 0;               // used internally by Ragel

    %% write init;             // initialize state machine

    len = fread( p, 1, space, stdin);
    pe = p + len;
    eof = pe;

    %% write exec;         // run state machine
}
```

Simplified scanner() function

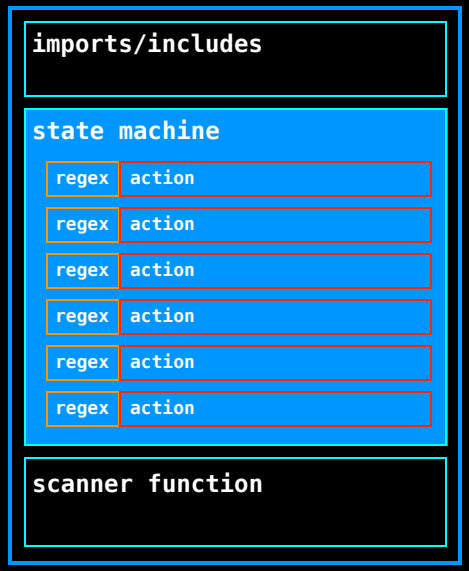
```
void scanner()
{
    // internal scanner( variables
    int len, space = BUFSIZE, curline = 1;
    static char buf[BUFSIZE];  // assume big enough for any input

    // Variables used for communicating with Ragel
    char *p = buf;              // pointer to the character data to process
    char *pe;                   // pointer to the end of the data to process
    char *eof;                  // same as pe unless using buffered input
    char *ts, *te = 0;          // start of token, end of token
    // Variables used internally by Ragel
    int cs = 0;                 // current state
    int act = 0;                // used internally by Ragel

    %% wri

    len =
    pe = p + len;
    eof = pe;

    %% write exec;          // run state machine
}
```

The *write exec* directive causes Ragel to emit the state machine's execution code, which may be hundreds or thousands of lines of inline C code.

Code Generation Sample



Setting up the State Machine



Recognizing an Identifier Token

- An identifier token:

  Starts with an alphabetic character or underscore, followed by zero or more alphanumeric or underscore characters

- Examples:

  Name
  Zarlon63
  _zodiac_kayak

Recognizing an Identifier Token

```
# Alpha numeric characters or underscore.
alnum_u = alnum | '_';

# Alpha characters or underscore.
alpha_u = alpha | '_';

# Identifier
alpha_u alnum_u* {
    printf( "ident(%i): ", curline );
    fwrite( ts, 1, te-ts, stdout );
    printf("\n");
};
```

Recognizing an Identifier Token

```
# Alpha numeric characters or underscore.
alnum_u = alnum | '_';

# Alpha characters or underscore.
alpha_u = alpha | '_';

# Identifier
alpha_u alnum_u* {
    printf( "ident(%i): ", curline );
    fwrite( ts, 1, te-ts, stdout );
    printf("\n");
};
```

First character of an identifier must be alphabetic or an underscore

Recognizing an Identifier Token

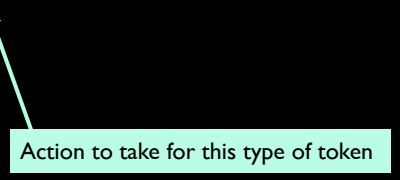
```
# Alpha numeric characters or underscore.
alnum_u = alnum | '_';

# Alpha characters or underscore.
alpha_u = alpha | '_';

# Identifier
alpha_u alnum_u* {
    printf( "ident(%i): ", curline );
    fwrite( ts, te-ts, stdout );
    printf("\n");
};
```

Repeat 0 or more times

Next character must be alphanumeric or an underscore

Processing a Token

```
# Alpha numeric characters or underscore.
alnum_u = alnum | '_';

# Alpha characters or underscore.
alpha_u = alpha | '_';

# Identifier
alpha_u alnum_u* {
    printf( "ident(%i): ", curline );
    fwrite( ts, 1, te-ts, stdout );
    printf("\n");
};
```



Processing a Token

```
# Alpha numeric characters or underscore.
alnum_u = alnum | '_';

# Alpha characters or underscore.
alpha_u = alpha | '_';

# Identifier
alpha_u alnum_u* {
    printf( "ident(%i): ", curline );
    fwrite( ts, 1, te-ts, stdout );
    printf("\n");
};
```

Print “ident” + line number

Processing a Token

```
# Alpha numberic characters or underscore.
alnum_u = alnum | '_';

# Alpha characters or underscore.
alpha_u = alpha | '_';

# Identif
alpha_u alnum_u* {
    printf( "ident(%i): ", curline );
    fwrite( ts, 1, te-ts, stdout );
    printf("\n");
};
```

Print identifier token

Length of token

Pointer to start of token

Mini-C Scanner Operation

Input:

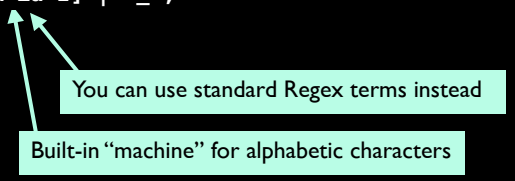
```
( Fahrenheit - 32 ) * 5 / 9
```

Output:

```
symbol(1):(
ident(1):Fahrenheit
symbol(1):-
int(1):32
symbol(1):)
symbol(1):*
int(1):5
symbol(1):/
int(1):9
```

Built in Regex Expressions

```
# Alpha numberic characters or underscore.
alnum_u = alnum | '_';

# Alpha characters or underscore.
alpha_u = [A-Za-z] | '_';
```



Built in Regex Expressions

any	*any character in the alphabet*	
alpha	*alphabetic characters*	`[A-Za-z]`
digit	*numeric digits*	`[0-9]`
alnum	*alphanumeric characters*	`[A-Za-z0-9]`
lower	*lower case alphabetic characters*	`[a-z]`
upper	*upper case alphabetic characters*	`[A-Z]`
xdigit	*hexadecimal digits*	`[0-9A-Fa-f]`
cntrl	*control characters*	`0..31`
graph	*graphical characters*	`[!-~]`
print	*printable characters*	`[ -~]`
punct	*punctuation characters*	`[!-/:-@[-‘{-~]`
space	*whitespace*	`[\t\v\f\n\r ]`
zlen	*zero length*	“”
empty	*empty set (matches nothing)*	`^any`

Terminology

Ragel calls regular expressions “machines”

Regex Learning Resources

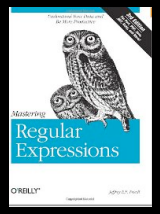
1) Online

http://www.regular-expressions.info/tutorial.html

http://en.wikipedia.org/wiki/Regular_expression

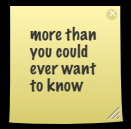
http://perldoc.perl.org/perlretut.html

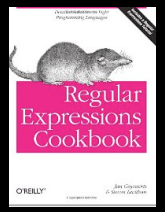
2) Books



Mastering Regular Expressions

Jeffery E.F. Friedl





Regular Expressions Cookbook

Jan Goyvaerts, Steven Levithan

Ragel, Unicode & Objective-C

Classes

- RGExpression
- ExpressionToken

RGExpression Class

Instance Variables

- Text
- Tokens (NSMutableArray)

Methods

- Convert text to tokens (calls Ragel)
- Add token to array (called from Ragel)
- Evaluate expression value

ExpressionToken Class

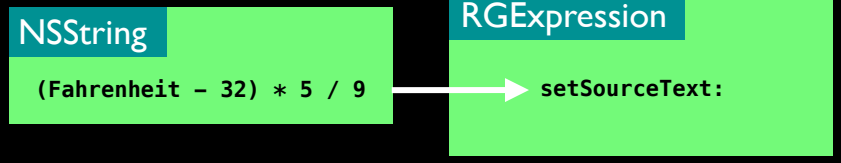
Instance Variables

- Text
- Value (type depends on subclass)

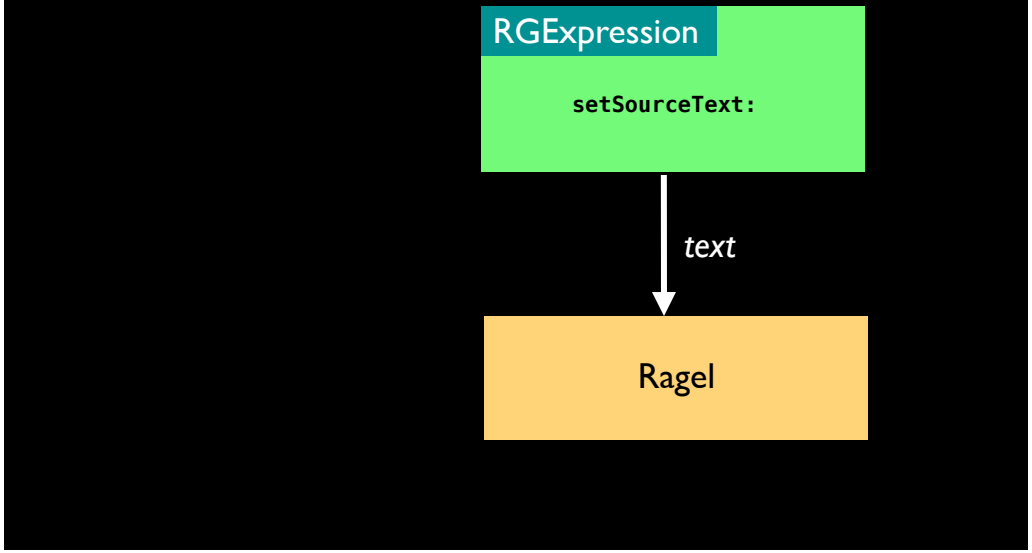
Subclasses

- IdentifierToken
- StringToken
- IntegerToken
- FloatToken

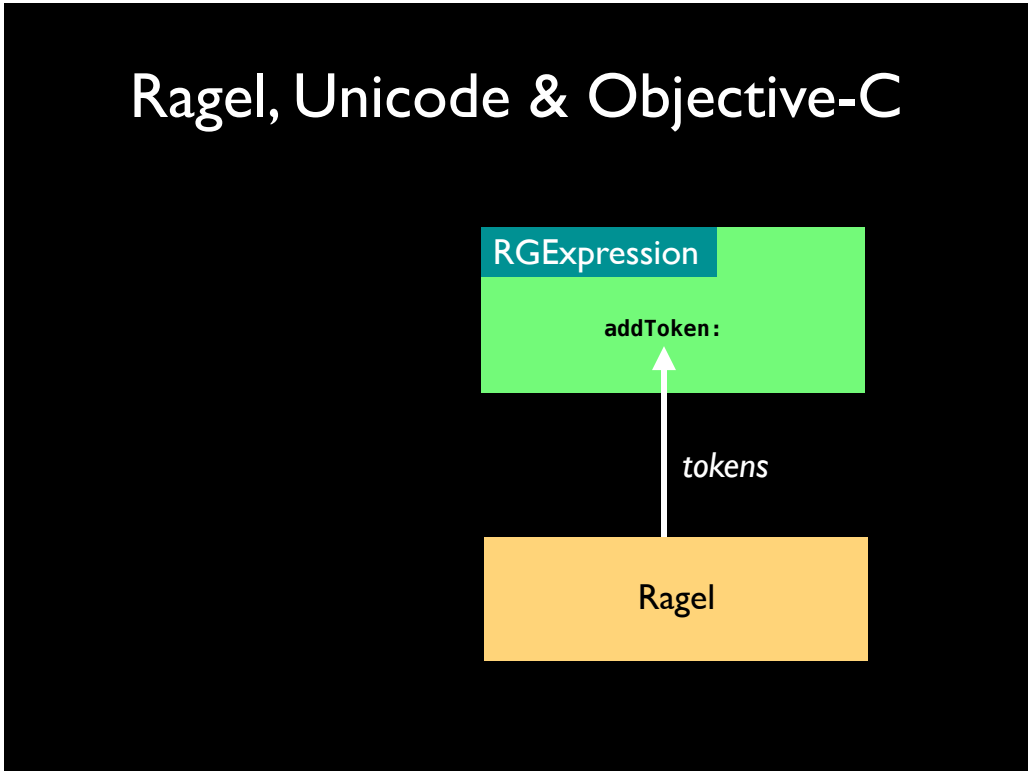
Ragel, Unicode & Objective-C



Ragel, Unicode & Objective-C



Ragel, Unicode & Objective-C



RGExpression Class

```
@interface RgExpression : NSObject {
    NSString * sourceText;
    NSData * sourceUnicode;
    NSMutableArray * tokens;
}

@implementation RgExpression

- (void) setSourceText:(NSString *) text
{
    sourceText = [text copy];
    sourceUnicode =
        [sourceText dataUsingEncoding:NSUTF16LittleEndianStringEncoding];
    short * sourceBase = (short *)[sourceUnicode bytes];
    unsigned long sourceLen = [sourceUnicode length]/2;
    tokens = [[NSMutableArray alloc] init];
    RgExpressionScanner(self, sourceBase, sourceLen);
}
```

RGExpression Class

```
@interface RgExpression : NSObject {
    NSString * sourceText;
    NSData * sourceUnicode;
    NSMutableArray * tokens;
}

@implementation RgExpression

- (void) setSourceText:(NSString *) text
{
    sourceText = [text copy];
    sourceUnicode =
        [sourceText dataUsingEncoding:NSUTF16LittleEndianStringEncoding];
    short * sourceBase = (short *)[sourceUnicode bytes];
    unsigned long sourceLen = [sourceUnicode length]/2;
    tokens = [[NSMutableArray alloc] init];
    RgExpressionScanner(self, sourceBase, sourceLen);
}
```

RGExpression Class

```
@interface RgExpression : NSObject {
    NSString * sourceText;
    NSData * sourceUnicode;
    NSMutableArray * tokens;
}

@implementation RgExpression

- (void) setSourceText:(NSString *) text
{
    sourceText = [text copy];
    sourceUnicode =
        [sourceText dataUsingEncoding:NSUTF16LittleEndianStringEncoding];
    short * sourceBase = (short *)[sourceUnicode bytes];
    unsigned long sourceLen = [sourceUnicode length]/2;
    tokens = [[NSMutableArray alloc] init];
    RgExpressionScanner(self, sourceBase, sourceLen);
}
```

RGExpression Class

```
@interface RgExpression : NSObject {
    NSString * sourceText;
    NSData * sourceUnicode;
    NSMutableArray * tokens;
}

@implementation RgExpression

- (void) setSourceText:(NSString *) text
{
    sourceText = [text copy];
    sourceUnicode =
        [sourceText dataUsingEncoding:NSUTF16LittleEndianStringEncoding];
    short * sourceBase = (short *)[sourceUnicode bytes];
    unsigned long sourceLen = [sourceUnicode length]/2;
    tokens = [[NSMutableArray alloc] init];
    RgExpressionScanner(self, sourceBase, sourceLen);
}
```

RGExpression Class

```
@interface RgExpression : NSObject {
    NSString * sourceText;
    NSData * sourceUnicode;
    NSMutableArray * tokens;
}

@implementation RgExpression

- (void) setSourceText:(NSString *) text
{
    sourceText = [text copy];
    sourceUnicode =
        [sourceText da                    ding:NSUTF16LittleEndianStringEncoding];
    short * sourceBase =  (short *)[sourceUnicode bytes];
    unsigned long sourceLen = [sourceUnicode length]/2;
    tokens = [[NSMutableArray alloc] init];
    RgExpressionScanner(self, sourceBase, sourceLen);
}
```

For callbacks

Number of characters

Pointer to UTF-16

Ragel Scanner

```
%%{
    machine clang;
    alphtype short;

    ... state machine definitions ...

}%%

%% write data nofinal;

void RgExpressionScanner(PanExpression *px, short *p, unsigned long len);
void RgExpressionScanner(PanExpression *px, short *p, unsigned long len)
{
    short *pe, *eof;
     int cs, act, curline = 1;
     short *ts, *te = 0;

     %% write init;

    pe = p+len;
    eof = pe;

    %% write exec;
}
```

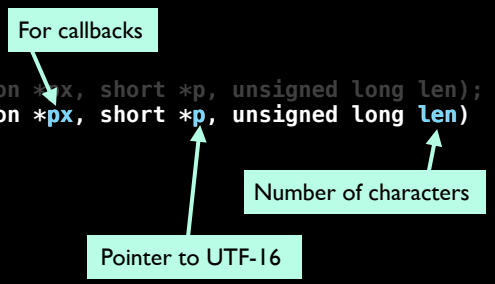
Ragel Scanner

```
%%{
    machine clang;
    alphtype short;

    ... state machine definitions ...

}%%

%% write data nofinal;

void RgExpressionScanner(PanExpression *px, short *p, unsigned long len);
void RgExpressionScanner(PanExpression *px, short *p, unsigned long len)
{
    short *pe, *eof;
     int cs, act, curline = 1;
     short *ts, *te = 0;

     %% write init;

    pe = p+len;
    eof = pe;

    %% write exec;
}
```

Tells Ragel to use UTF-16 instead of ASCII

Ragel Scanner

```
%%{
    machine clang;
    alphtype short;

    ... state machine definitions ...

}%%

%% write data nofinal;

void RgExpressionScanner(PanExpression *px, short *p, unsigned long len);
void RgExpressionScanner(PanExpression *px, short *p, unsigned long len)
{
    short *pe, *eof;
     int cs, act, curline = 1;
     short *ts, *te = 0;

     %% write init;

    pe = p+len;
    eof = pe;

    %% write exec;
}
```



Ragel Scanner

```
%%{
    machine clang;
    alphtype short;

    ... state machine definitions ...

}%%

%% write data nofinal;

void RgExpressionScanner(PanExpression *px, short *p, unsigned long len);
void RgExpressionScanner(PanExpression *px, short *p, unsigned long len)
{
    short *pe, *eof;
     int cs, act, curline = 1;
     short *ts, *te = 0;

     %% write init;

    pe = p+len;
    eof = pe;

    %% write exec;
}
```

Change pointers to short instead of char

Add Token to RGExpression

```
# Alpha numberic characters or underscore.
alnum_u = alnum | '_';

# Alpha characters or underscore.
alpha_u = alpha | '_';

# Identifier
alpha_u alnum_u* {
    [px addToken:[IdentifierToken alloc] start:ts end: te];
};
```

Add Token to RGExpression

```
# Alpha numberic characters or underscore.
alnum_u = alnum | '_';

# Alpha characters or underscore.
alpha_u = alpha | '_';

# Identifier
alpha_u alnum_u* {
    [px addToken:[IdentifierToken alloc] start:ts end: te];
};
```

Start of token (in UTF-16 source)

Create unitialized token object

End of token (in UTF-16 source)

Callback to RGExpression object

Add Token to RGExpression

```
- (void) addToken:(ExpressionToken *)token
            start:(short *)ts
              end:(short *)te
{
    short * sourceBase = (short *)[sourceUnicode bytes];
    long tokenSpot = ts-sourceBase;
    long tokenLength = te-ts;
    NSString * xtext = [[NSString alloc]
                        initWithBytes:(const void *)ts
                               length: tokenLength*2
                             encoding:NSUTF16LittleEndianStringEncoding];
    token = [token initWithExpression:self
                                Token:xtext
                                Start:tokenSpot
                                  Len:tokenLength];
    [tokens addObject:token];
}
```

Add Token to RGExpression

```
- (void) addToken:(ExpressionToken *)token
            start:(short *)ts
              end:(short *)te
{
    short * sourceBase = (short *)[sourceUnicode bytes];
    long tokenSpot = ts-sourceBase;
    long tokenLength = te-ts;
    NSString * xtext = [[NSString alloc]
                        initWithBytes:(const void *)ts
                               length: tokenLength*2
                             encoding:NSUTF16LittleEndianStringEncoding];
    token = [token initWithExpression:self
                                Token:xtext
                                Start:tokenSpot
                                  Len:tokenLength];
    [tokens addObject:token];
}
```

Unitialized token object

Add Token to RGExpression

```
- (void) addToken:(ExpressionToken *)token
            start:(short *)ts
              end:(short *)te
{
    short * sourceBase = (short *)[sourceUnicode bytes];
    long tokenSpot = ts-sourceBase;
    long tokenLength = te-ts;
    NSString * xtext = [[NSString alloc]
                        initWithBytes:(const void *)ts
                               length: tokenLength*2
                             encoding:NSUTF16LittleEndianStringEncoding];
    token = [token initWithExpression:self
                                Token:xtext
                                Start:tokenSpot
                                  Len:tokenLength];
    [tokens addObject:token];
}
```

Start of token (in UTF-16 source)

End of token (in UTF-16 source)

Add Token to RGExpression

```
- (void) addToken:(ExpressionToken *)token
            start:(short *)ts
              end:(short *)te
{
    short * sourceBase = (short *)[sourceUnicode bytes];
    long tokenSpot = ts-sourceBase;
    long tokenLength = te-ts;
    NSString * xtext = [[NSString alloc]
                        initWithBytes:(const void *)ts
                               length: tokenLength*2
                             encoding:NSUTF16LittleEndianStringEncoding];
    token = [token initWithExpression:self
                                Token:xtext
                                Start:tokenSpot
                                  Len:tokenLength];
    [tokens addObject:token];
}
```

String value of token

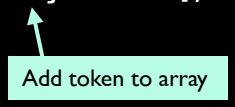
Add Token to RGExpression

```
- (void) addToken:(ExpressionToken *)token
            start:(short *)ts
              end:(short *)te
{
    short * sourceBase = (short *)[sourceUnicode bytes];
    long tokenSpot = ts-sourceBase;
    long tokenLength = te-ts;
    NSString * xtext = [[NSString alloc]
                        initWithBytes:(const void *)ts
                               length: tokenLength*2
                             encoding:NSUTF16LittleEndianStringEncoding];
    token = [token initWithExpression:self
                                Token:xtext
                                Start:tokenSpot
                                  Len:tokenLength];

    [tokens addObject:token];
}
```

Initialize the token

Add Token to RGExpression

```
- (void) addToken:(ExpressionToken *)token
            start:(short *)ts
              end:(short *)te
{
    short * sourceBase = (short *)[sourceUnicode bytes];
    long tokenSpot = ts-sourceBase;
    long tokenLength = te-ts;
    NSString * xtext = [[NSString alloc]
                        initWithBytes:(const void *)ts
                               length: tokenLength*2
                             encoding:NSUTF16LittleEndianStringEncoding];
    token = [token initWithExpression:self
                                Token:xtext
                                Start:tokenSpot
                                  Len:tokenLength];

    [tokens addObject:token];
}
```

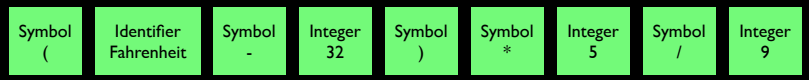


Typical Operation

Input (NSString):

```
( Fahrenheit – 32 ) * 5 / 9
```

Output (NSMutableArray):



Scanning for Unicode Characters

```
# Pi (Option-P)
π | ∏ {
    [px addToken:[FloatOperandToken alloc] start:ts end: te];
};
```

Won't work, Ragel does not allow Unicode in source

Scanning for Unicode Characters

```
# Pi (Option-P)
0x3C0 | 0x220F {
    [px addToken:[FloatOperandToken alloc] start:ts end: te];
};
```

Scanning for Unicode Characters

π Π

```
# Pi (Option-P)
0x3C0 | 0x220F {
    [px addToken:[FloatOperandToken alloc] start:ts end: te];
};
```

Scanning for Unicode Characters

```
# Pi (Option-P)
0x3C0 | 0x220F {
    [px addToken:[FloatOperandToken alloc] start:ts end: te];
};
```

Token Initialization

```
@implementation FloatOperandToken
@synthesize value;

- (id) initWithExpression:(PanExpression *)xpr
                    Token:(NSString *)token
                    Start:(long)start
                      Len:(long)len;
{
    if (self = [super initWithExpression:xpr
                                   Token:token Start:start Len:len]) {
        dataType = FLOAT;
        NSString *text =
            [[expression getSourceText] substringWithRange:tokenStartEnd];
        if ([text isEqualToString:@"π"] || [text isEqualToString:@"∏"]) {
            value = M_PI;
        } else {
            value = [text doubleValue];
        }
    }
    return self;
}
```

Token Initialization

```
@implementation FloatOperandToken
@synthesize value;

- (id) initWithExpression:(PanExpression *)xpr
                    Token:(NSString *)token
                    Start:(long)start
                      Len:(long)len;
{
    if (self = [super initWithExpression:xpr
                                   Token:token Start:start Len:len]) {
        dataType = FLOAT;
        NSString *text =
            [[expression getSourceText] substringWithRange:tokenStartEnd];
        if ([text isEqualToString:@"π"] || [text isEqualToString:@"∏"]) {
            value = M_PI;
        } else {
            value = [text doubleValue];
        }
    }
    return self;
}
```

Token Initialization

```
@implementation FloatOperandToken
@synthesize value;

- (id) initWithExpression:(PanExpression *)xpr
                    Token:(NSString *)token
                    Start:(long)start
                      Len:(long)len;
{
    if (self = [super initWithExpression:xpr
                                   Token:token Start:start Len:len]) {
        dataType = FLOAT;
        NSString *text =
            [[expression getSourceText] substringWithRange:tokenStartEnd];
        if ([text isEqualToString:@"π"] || [text isEqualToString:@"∏"]) {
            value = M_PI;
        } else {
            value = [text doubleValue];
        }
    }
    return self;
}
```

Token Initialization

```
@implementation FloatOperandToken
@synthesize value;

- (id) initWithExpression:(PanExpression *)xpr
                    Token:(NSString *)token
                    Start:(long)start
                      Len:(long)len;
{
    if (self = [super initWithExpression:xpr
                                   Token:token Start:start Len:len]) {
        dataType = FLOAT;
        NSString *text =
            [[expression getSourceText] substringWithRange:tokenStartEnd];
        if ([text isEqualToString:@"π"] || [text isEqualToString:@"∏"]) {
            value = M_PI;
        } else {
            value = [text doubleValue];
        }
    }
    return self;
}
```

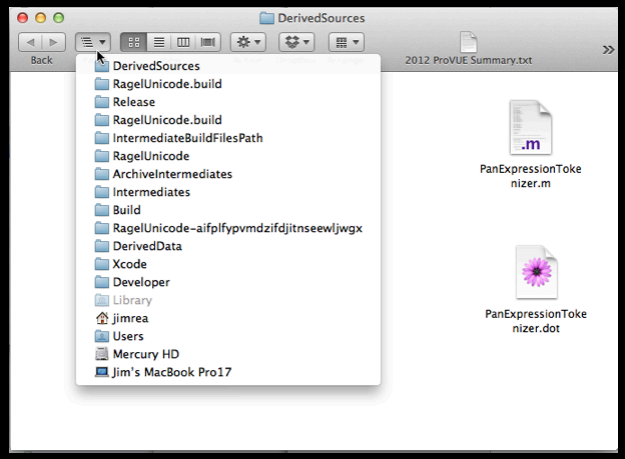
Token Initialization

```
@implementation FloatOperandToken
@synthesize value;

- (id) initWithExpression:(PanExpression *)xpr
                    Token:(NSString *)token
                    Start:(long)start
                      Len:(long)len;
{
    if (self = [super initWithExpression:xpr
                                   Token:token Start:start Len:len]) {
        dataType = FLOAT;
        NSString *text =
            [[expression getSourceText] substringWithRange:tokenStartEnd];
        if ([text isEqualToString:@"π"] || [text isEqualToString:@"∏"]) {
            value = M_PI;
        } else {
            value = [text doubleValue];
        }
    }
    return self;
}
```

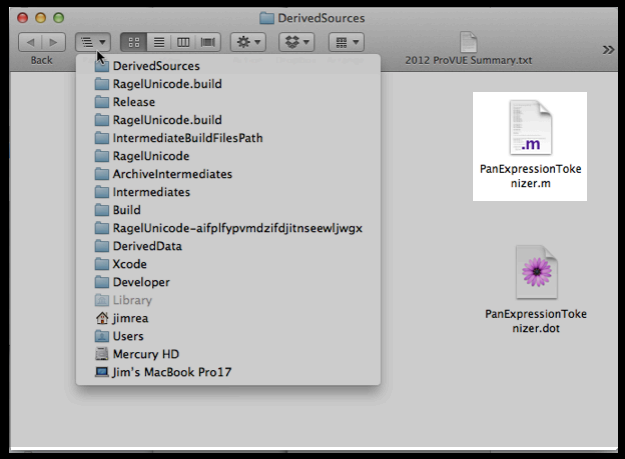
Floating Point Constant

```
# Floating Point Number
(digit+ '.' digit+ ('e' ('-'|'+')? digit+)?) | (digit+ 'e' ('-'|'+')?
digit+) {
    [px addToken:[FloatOperandToken alloc] start:ts end: te];
};
```

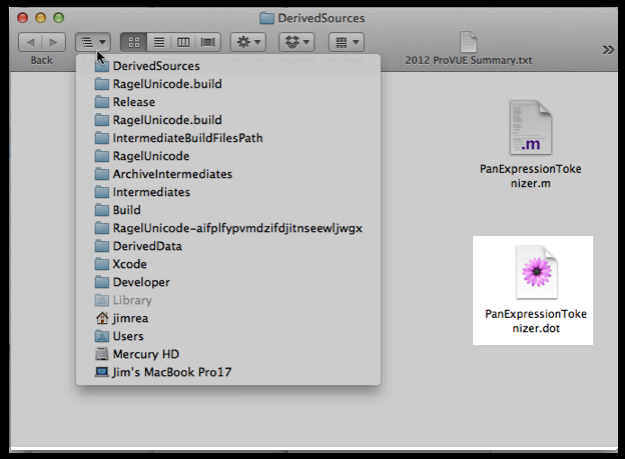
Ragel Outputs



Ragel Outputs



Ragel Outputs

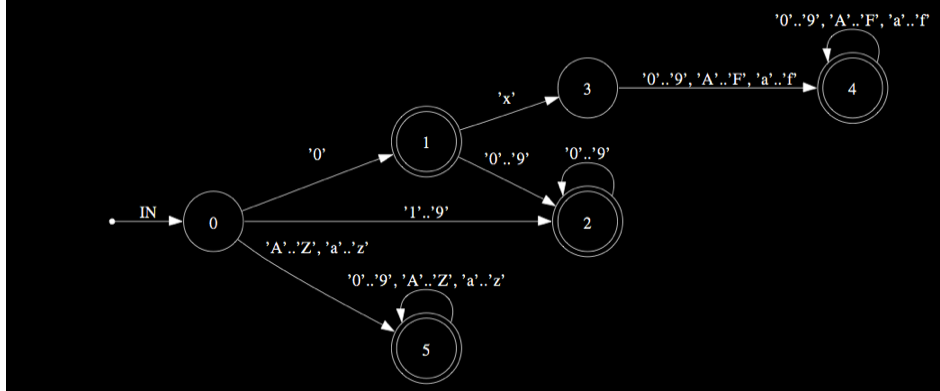


GraphViz

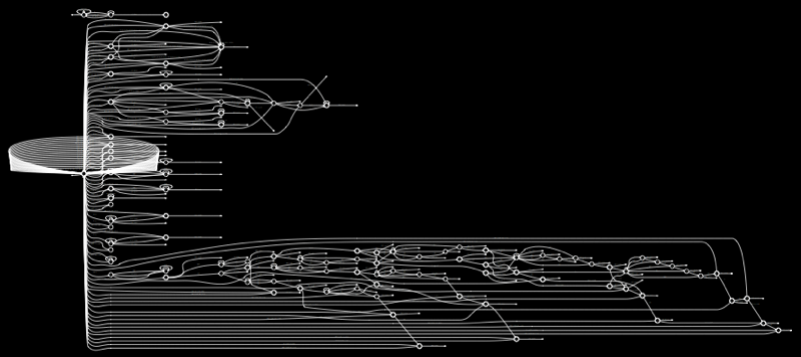
Open source graph visualization software

http://www.graphviz.org

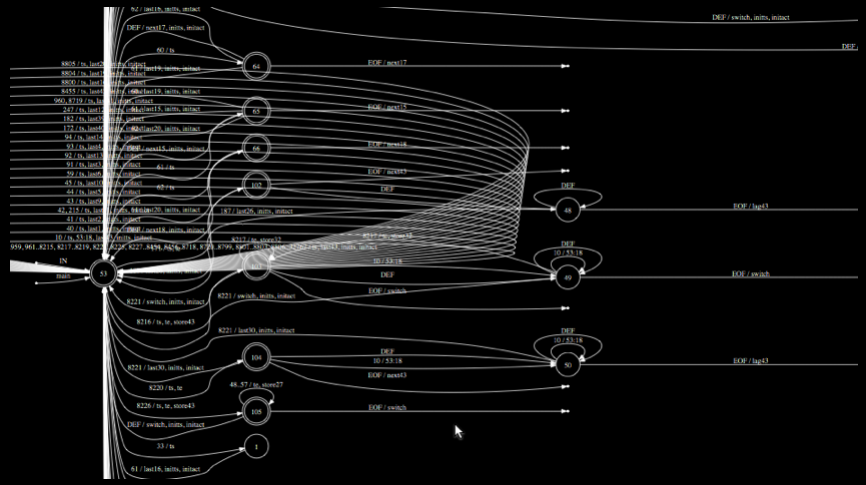
GraphViz



GraphViz



GraphViz


DEF / switch, initts, initact
EOF / next17
EOF / next15
EOF / next16
EOF / next43
EOF / lag43
EOF / switch
EOF / lag43
EOF / next43
EOF / switch
DEF
IN
start
53
64
65
66
102
48
49
50
104
105
1

Resources

1) Ragel Home Page

http://www.complang.org/ragel/

2) Ragel User Guide

http://www.complang.org/ragel/ragel-guide-6.6.pdf

3) My Google+ post on using Ragel with Xcode

https://plus.google.com/109028627294998653069/posts/C6d1YkcY8kW

Credits

Ragel was written by Dr. Adrian Thurston, a software researcher at a Canadian network security firm. It was originally developed in early 2000 and was first released January 2002. Many people have contributed feedback, ideas and code.