Swift Fox reference manual

Marcin Szczodrak [*], Vasileios P. Kemerlis [†]

Xuan Linh Vu [‡] and Yiwei Gu [§]

Computer Science Department
Columbia University
New York, NY

Manuscript received at March 24, 2010; revised September 30, 2011

1 Introduction

This manual describes the first draft of Swift Fox language as specified by the authors on September 30, 2011. In the following sections we define in detail the basic constructs used in Swift Fox. Special consideration have been given in making this manual reader-friendly and comprehensible, but at the same time formal enough so as to capture the the basic concepts that Swift Fox employs.

> Commentary material is indented and written in smaller type, similar to this excerpt.

2 Lexical conventions

In the Swift Fox language, a program consists of one or more *translation units* stored into different files. The translation process consists of multiple phases, which are described in great detail in [7]. In the following section, we

[*] msz@cs.columbia.edu
[†] vpk@cs.columbia.edu
[‡] xv2103@columbia.edu
[§] yg2181@columbia.edu

present the primitive constructs that are used throughout the first phase of the translation, namely the *lexical analysis*. During that phase, the source code of a Swift Fox program is considered to be a stream of characters that is fed into the lexical analyzer (lexer). Subsequently, the lexer[1] groups sequences of characters together and identifies *tokens*. Each token is a pair of a `name` and a `value`; the value corresponds to a particular *lexeme* that is identified by the regular expression for the corresponding token, whereas the name is merely an identifier for abstracting the set of lexemes (*i.e.,* referring to the language of lexemes that are described by the same regular expression) [1].

token: $<$ `token_name`, `token_value` $>$ (*e.g.,* $<$ `identifier`, `blink` $>$)

> Appendix A includes the definition of a Swift Fox lexical analyzer for the `flex` lexer-generator [4, 6]. The reference implementation of the Swift Fox compiler uses the `flex` lexer-generator and the `YACC` parser-generator, [3, 5], however this manuscript provides (hopefully) all possible details for implementing a Swift Fox compiler in every available compiler-compiler tool.

2.1 Tokens

Tokens in Swift Fox, similarly to every other language, are composed by sequences of characters. The *lexer* (*i.e.,* lexical analyzer) produces four classes of tokens:

- `identifiers`
- `keywords`
- `constants`
- `operators`

Additionally, the *whitespace* character class is used in order to separate the various tokens. The following characters are considered whitespace and along with comments are ignored throughout the rest phases of the translation.

- `space`

[1] Also known as *scanner*.

- `horizontal tab`
- `vertical tab`
- `form feed`
- `carriage return`

Notice, however, that `newlines` are not considered whitespace and they have a special meaning in Swift Fox (see Section 3.3).

> Swift Fox is currently under development. Therefore, there might be some inconsistency between the formal description of the tokens and the corresponding `Lex` definition in Appendix A.

2.2 Comments

Comments are introduced by character `#` and terminate with a newline. Everything between the comment character and the newline character is considered to be a comment, and therefore it is ignored. Comments must start on a new line, and can only be preceded by a delimiter, but not a line of a Swift Fox code.

2.3 Identifiers

Identifiers in Swift Fox are sequences of letters, digits, and character "_". They are used in order to *name* specific instances of the primitive types that the language provides (see Section 3.1) and, hence, they are considered to be variables. Identifiers can be of arbitrary length but not zero (*i.e.,* there is no such thing as the "empty" identifier). Moreover, the first character of an identifier cannot be a number; it must be either a letter, or character "_". Finally, identifiers are case sensitive and have a number of characters that are considered to be significant.

> Notice that the number of significant characters in identifiers depends on the compiler implementation. The reference implementation provided by the authors supports up to 127 significant characters.

Identifiers in Swift Fox are only used for naming objects (*e.g.,* applications, networks, sources, configurations, event-conditions) and referring to

them. The binding between a name and the corresponding object is performed during the declaration of the identifier and remains the same for the rest of the program. For example, Swift Fox does not allow a user to change a configuration variable and assign a different configuration to it. The *scope* of all identifiers is, therefore, the same and it global (for a particular program). Similarly, the *lifetime* of all identifiers is the same and it is equal to the lifetime of the Swift Fox program.

> Objects in Swift Fox are not created and disposed *dynamically* (or *automatically*) at runtime.

2.4 Keywords

Swift Fox uses a set of special-meaning identifiers, namely *keywords*, which cannot be used otherwise. Table 1, below, illustrates all the "reserved" words that are used in Swift Fox.

`use`	`application`	`network`
`source`	`configuration`	`event-condition`
`from`	`goto`	`when`
`nothing`	`any`	`once`
`and`	`or`	`not`
	`start`	

Table 1: Swift Fox keywords.

> All keywords have special meaning for the Swift Fox translation procedure and therefore cannot be used as identifiers; one cannot name a configuration or an event-condition as "network", or "goto".

2.5 Constants

Currently only four different kinds of *constants* are supported. Temperature, time, integer constants, and string literals. Temperature and time constants are comprised of digit sequences that resemble a decimal value, followed by the corresponding suffix that denotes the scale. The allowed suffixes for temperature are "`C`" and "`F`" and they denote the Celsius and Fahrenheit scale respectively. Similarly, the time-allowed suffixes are "`msec`", "`sec`",

"`min`", "`hr`". On the other hand, integer constants are not allowed to have a suffix; they are merely comprised of digit sequences that resemble a decimal value. Finally, Swift Fox also supports string literals, which are essentially sequences of characters.

> Typically, string constants are used in order to denote path elements on the definition of applications, network protocols, and sources. They are semantically compatible with the definition of C strings.

3 Syntax notation

In this section we define the syntactic constructs that are used in Swift Fox language. The *syntactic analysis* is the bulk part of the second phase of the translation procedure [1]. During that phase, a stream of tokens, which is provided by the lexer, is checked for adherence to Swift Fox rules. In other words, the source code is tested for conformity to the syntactic rules that are defined by the formal grammar of the language. The result of this procedure is an *abstract syntax tree* (AST) of the program along with a special data structure that keeps information about each identifier (*i.e.,* the *symbol table*).

> Appendix B includes the definition of a Swift Fox parser. The reference implementation of the Swift Fox compiler uses the YACC (Bison) parser-generator [3, 5]. However, this manuscript provides all possible details for implementing a Swift Fox compiler with every available compiler-compiler.

3.1 Types

Swift Fox is a *strongly typed* language [2] where type checking occurs only at compile time. The simplistic nature of Swift Fox syntax prevents type intermixing without loss of expressibility. The benefits of this approach are manifold. First, the language does not need to involve additional complexity with runtime checks. Notice that the execution environment of Swift Fox is severely constrained and any runtime type check will not only impact the performance of the system, but also drain the available power supply [8]. Second, it allows strong guarantees about the runtime behavior of the program since there are no explicit or implicit type conversions. Third, it

guards against evasions of the type system that can lead to unpredictable behavior.

Swift Fox is also a *static typed* language. This allows the optimal selection of the storage needed for the various Swift Fox objects. This is, again, of paramount importance for the over constrained application domain of Swift Fox [8]. The *primitive* types that define the building blocks of the language are the following:

- `application`. This primitive type is used in order to define application components (*e.g.,* Blink, Collector, or Picture) as introduced in [7]
- `network`. Similarly to the previous type, network is used in order to define network components (*e.g.,* CTP or P2P-MultiHop)
- `source`. Defines event sources (*e.g.,* temperature readings, timeouts)
- `configuration`. Configuration defines a specific binding among instances of the different classes of components that are provided by Fennec Fox [7])
- `event-condition`. Event-condition associates events with specific conditions (*e.g.,* temperature $>$ 90 F)

> Swift Fox does not have composite types nor does it allow the programmer to define and use its own types. Moreover, it provides a set of predefined values for application, network, and source that correspond to the basic applications, network protocols, and sensors that are always available to a system. This feature is essentially similar to the notion of C libraries that provide additional functionality in user programs when needed.

3.2 Operators

The operators used in Swift Fox can be classified as *relational*, *logical*, or *enumerative*. Relational operators are used in order to define an event-condition and they evaluate to `true` or `false`. Infix operators `<` (less), `>` (greater), `<=` (less or equal), `>=` (greater or equal), and `=` (equal) yield 0, if the relation that corresponds to a specific condition is false, and 1 if it is true. The infix logical operators `and` and `or` are used in order to combine two or more event-conditions conjunctively or disjunctively, whereas the

infix *comma* (`,`) operator is used among two or more configurations in order to create a set from them. Finally, the logical negation operator `not` is followed by an event-condition. The result of the negation is 1 when the event-condition evaluates to true, and 0 otherwise.

3.3 Separators

Separators are special characters that are used in Swift Fox in order to segregate various statements. Currently, Swift Fox uses only one separator: the `newline` (`LF`). Hence, library declaration statements, configuration and event-condition declaration statements, and policy and initial-configuration statements (Section 3.4 and Appendix B) are separated from each other using newline characters.

3.4 Statements

Swift Fox programs consist of sequences of *statements*. There are three different types of *statements*. The first type includes *declaration statements* for applications, network protocols, and event sources, as well as configurations and event-conditions (see Section 3.1). The lines 154 – 231 and 394 – 453 in Appendix B illustrate the syntax for that particular type. The second type of statements is about *policy definitions* (see lines 252 – 380 in Appendix B). The bulk part of a Swift Fox program is typically made of such statements, since they capture the reconfiguration strategy of the system. Finally, the last type of statement (actually it is only one statement) is about declaring the initial configuration of the system; similar to the main entry point of a C program, there is an initial configuration for a Swift Fox policy.

> Appendix B illustrates the syntax of the Swift Fox language through a YACC grammar-definition. Without loss of generality, this should give an indication of the syntactic rules of the language so as to implement those in other compiler-compilers.

Notice that the order of the previous statements is important and fixed. A valid Swift Fox program cannot intermix different types of statements and the first statements of a valid program should be application, network, or source declarations followed by configuration declarations. The application, network, and source declarations are optional (*i.e.,* they can be omitted) and are typically provided in the form of a "library". Subsequently, there

might be some event-condition declaration statements followed by policy statements. Again, the declaration of event-conditions and the definition of policies is not mandatory. The final statement is always the initial configuration statement and it is necessary in every valid Swift Fox program.

A Swift Fox Lex definition

```
%{
/* Swift Fox Compiler v0.3
 * Authors: Marcin Szczodrak and Vasileios P. Kemerlis
 * Date: May 9, 2010
 */

#include <stdio.h>
#ifdef __DEBUG__          /* link with the testing suite */
#include <unistd.h>
#include "common.h"
#else                     /* link with YACC/Bison */
#include <fcntl.h>
#include <ctype.h>
#include <stdlib.h>
#include "y.tab.h"
#include "sf.h"

#define YY_NO_INPUT
#endif

%}

%option nounput

delim             [ \v\f\r]
tab               \t+
whitespace        {delim}+
letter            [A-Za-z]
digit             [0-9]
number            [1-9]{digit}*
newline           \n.*
identifier        ({letter}|_)({letter}|{digit}|_)*
module_param      {digit}|,|{whitespace}
modules           {identifier}\({module_param}*\)
numtype           (C|F|msec|sec|min|hr)
constant          ({number}{numtype}?|ON|OFF)
env_const         \$\(({letter}|{digit}|_)*\)
path              (http:\/\/|\.|\/|{env_const})({letter}|{digit
    }|\.|\/|_|-|~)*
```

```
comment         #.*

%%

{newline}       {
#ifndef __DEBUG__
                (void)memset(linebuf, 0, BUF_SZ);
                strncpy(linebuf, yytext + 1, BUF_SZ - 1)
                    ; lineno++;
                tokenpos = 0; yyless(1);
#else
                yyless(1);
#endif
                return LF;
                    }
{tab}           {
#ifndef __DEBUG__
                tokenpos += ((yyleng*5) + (yyleng - 1) *
                     3);
#endif
                                                        }



{whitespace}    {
#ifndef __DEBUG__
                tokenpos += yyleng;
#endif
                                                }
{comment}       {
#ifndef __DEBUG__
                tokenpos = 0;
#endif
                                                }
any             {
#ifndef __DEBUG__
                tokenpos += yyleng;
                yylval.symp = symlook(yytext);
#endif
                return ANY;                     }
(configuration|conf)    {
#ifndef __DEBUG__
                tokenpos += yyleng;
#endif
                return CONFIGURATION;           }
,               {
#ifndef __DEBUG__
                tokenpos += yyleng;
#endif
```

```
                        return COMMA;                   }

(nothing|none)  {
#ifndef __DEBUG__
                        tokenpos += yyleng;
                        yylval.symp = symlook(yytext);
#endif
                        return NOTHING;                 }
(event-condition|event) {
#ifndef __DEBUG__
                        tokenpos += yyleng;
#endif
                        return EVENT_CONDITION;         }
from            {
#ifndef __DEBUG__
                        tokenpos += yyleng;
#endif
                        return FROM;                    }
goto            {
#ifndef __DEBUG__
                        tokenpos += yyleng;
#endif
                        return GOTO;                    }
start           {
#ifndef __DEBUG__
                        tokenpos += yyleng;
#endif
                        return START;                   }
use             {
#ifndef __DEBUG__
                        tokenpos += yyleng;
#endif
                        return  USE;                    }
application     {
#ifndef __DEBUG__
                        tokenpos += yyleng;
#endif
                        return APPLICATION;             }
network         {
#ifndef __DEBUG__
                        tokenpos += yyleng;
#endif
                        return NETWORK;                 }
qoi         {
#ifndef __DEBUG__
                        tokenpos += yyleng;
#endif
                        return QOI;                 }
mac         {
```

```
#ifndef __DEBUG__
                        tokenpos += yyleng;
#endif
                        return MAC;                     }
radio           {
#ifndef __DEBUG__
                        tokenpos += yyleng;
#endif
                        return RADIO;                   }
address         {
#ifndef __DEBUG__
                        tokenpos += yyleng;
#endif
                        return ADDRESS;                 }
source          {
#ifndef __DEBUG__
                        tokenpos += yyleng;
#endif
                        return SOURCE;                  }
once            {
#ifndef __DEBUG__
                        tokenpos += yyleng;
#endif
                        return ONCE;                    }
when            {
#ifndef __DEBUG__
                        tokenpos += yyleng;
#endif
                        return WHEN;                    }
and             {
#ifndef __DEBUG__
                        tokenpos += yyleng;
                        yylval.ival = AND;
#endif
                        return AND;                     }
or              {
#ifndef __DEBUG__
                        tokenpos += yyleng;
                        yylval.ival = OR;
#endif
                        return OR;                      }
not             {
#ifndef __DEBUG__
                        tokenpos += yyleng;
                        yylval.ival = NOT;
#endif
                        return NOT;                     }
"<"             {
#ifndef __DEBUG__
```

```
                        tokenpos += yyleng;
                        yylval.ival = LT;
#endif
                        return RELOP;                   }
">"             {
#ifndef __DEBUG__
                        tokenpos += yyleng;
                        yylval.ival = GT;
#endif
                        return RELOP;                   }
"<="            {
#ifndef __DEBUG__
                        tokenpos += yyleng;
                        yylval.ival = LE;
#endif
                        return RELOP;                   }
">="            {
#ifndef __DEBUG__
                        tokenpos += yyleng;
                        yylval.ival = GE;
#endif
                        return RELOP;                   }
"<>"            {
#ifndef __DEBUG__
                        tokenpos += yyleng;
                        yylval.ival = NE;
#endif
                        return RELOP;                   }
"="             {
#ifndef __DEBUG__
                        tokenpos += yyleng;
                        yylval.ival = EQ;
#endif
                        return RELOP;                   }
\{              {
#ifndef __DEBUG__
                        tokenpos += yyleng;
#endif
                        return OPEN_BRACE;              }
\}              {
#ifndef __DEBUG__
                        tokenpos += yyleng;
#endif
                        return CLOSE_BRACE;             }
{identifier}    {
#ifndef __DEBUG__
                        tokenpos += yyleng;
                        yylval.symp = symlook(yytext);
#endif
```

```
231                             return IDENTIFIER;              }

    {modules}     {
#ifndef __DEBUG__
                                tokenpos += yyleng;
236                             yylval.modp = proc_module(yytext);
#endif
                                return MODULES;                 }


241 {path}            {
#ifndef __DEBUG__
                                tokenpos += yyleng;
                                yylval.libp = liblook(yytext);
#endif
246                             return PATH;                    }
    {constant}        {
#ifndef __DEBUG__
                                tokenpos += yyleng;
                                yylval.symp = symlook(yytext);
251 #endif
                                return CONSTANT;                }

    virtual-network {
                                return VIRTUAL_NETWORK;         }
256
    %%

    #ifdef __DEBUG__            /* link with the testing suite */
    int
261 main(int argc, char **argv) {

            int tok;            /* token from the scanner */

            /* try to open the first argument for input */
266         if ((argc != 1) && ((yyin = fopen(argv[1], "r")) == NULL
                ))
                    /* failed */
                    yyin = stdin;   /* read from stdin */

            /* call the scanner repetitively */
271         while ((tok = yylex()) != 0)
                    /* print the identified token */
                    (void)fprintf(stdout, "%d ", tok);
            /* EOL */
            (void)fprintf(stdout, "\n");
276
            /* cleanup */
            (void)fclose(yyin);
```

```
        /* finish */
281     return EXIT_SUCCESS;
}
#endif
```

../../src/sf/sf.l

B Swift Fox YACC definition

```
%{
2 /* Swift Fox Compiler v0.3
 * Authors: Marcin Szczodrak and Vasileios P. Kemerlis
 * Date: May 9, 2010
 */

7 #include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <string.h>
#include <sys/stat.h>
12 #include <ctype.h>
#include "sf.h"
#include "traverse.h"

char *relopToLetter(int i);
17 int editConst(struct symtab *entry);
int negateOperator(int i);
void initialize(void);
void gc(void);
void checkForRemotePath(struct libtab*);
22
int conf_counter        = 1;
int event_counter       = 1;
int policy_counter      = 0;
int virtual_counter     = 0;
27 int event_id_counter    = 0;
FILE *fcode             = NULL;

int app_id_counter = 1;
int net_id_counter = 1;
32 int mac_id_counter = 1;
int qoi_id_counter = 1;
int addr_id_counter = 1;
int radio_id_counter = 1;

37 struct eventnodes *last_evens = NULL;
```

```
%}

%union {
        struct symtab           *symp;
        struct modtab           *modp;
        struct libtab           *libp;
        int                     ival;
        char                    *str;
        double                  dval;
        struct confnode         *confp;
        struct confnodes        *confsp;
        struct eventnode        *evep;
        struct eventnodes       *evesp;
        struct policy           *pol;
        struct policies         *pols;
        struct initnode         *initp;
        struct program          *prog;
}

%token CONFIGURATION COMMA EVENT_CONDITION
%token FROM GOTO START USE WHEN
%token APPLICATION NETWORK QOI MAC RADIO ADDRESS
%token SOURCE LF VIRTUAL_NETWORK
%token LT GT LE GE NE EQ
%token OPEN_BRACE CLOSE_BRACE ONCE

%token <symp>   CONSTANT
%token <symp>   IDENTIFIER
%token <libp>   PATH
%token <ival>   RELOP
%token <ival>   AND
%token <ival>   OR
%token <ival>   NOT
%token <modp>   NOTHING
%token <symp>   ANY
%token <modp>   MODULES
%type <modp>    conf_param
%type <symp>    from_configurations
%type <confp>   configuration
%type <confsp>  configurations
%type <confsp>  defined_configurations
%type <evep>    event_condition
%type <evesp>   defined_events
%type <pol>     policy;
%type <pols>    policies;
%type <initp>   initial_configuration
%type <prog>    program
%type <str>     type
```

```
%type <ival>     when_events
%type <ival>     one_event
%type <ival>     virtual_network
%type <ival>     virtual_networks
%type <symp>     configuration_ids

%%

swiftfox: library program
                {
                        /* verbose; to be removed */
                        /* printTable(); */

                        /* root node for the constructed AST */
                        struct program *p = $2;

                        /* traverse the AST for semantic
                            checking */
                        traverse_program($2,
                                        TREE_CHECK_SEMANTIC,
                                        policy_counter);

                        /* traverse the AST for code generation
                            */
                        traverse_program($2,
                                        TREE_GENERATE_CODE,
                                        policy_counter);
                }
        ;

program: defined_configurations defined_events policies
    virtual_networks initial_configuration
                {
                        /* root node */
                        $$              = calloc(1, sizeof(
                            struct program));

                        /* link the node appropriately */
                        $1->parent      = NULL;
                        if ($2 != NULL )
                                $2->parent = NULL;
                        if ($3 != NULL )
                                $3->parent = NULL;

                        /* init */
                        $$->defcon      = $1;
                        $$->defeve      = $2;
                        $$->defpol      = $3;
                        $$->init        = $5;
```

```
132                 }
            ;

    defined_configurations: configurations configuration
                {
137                     /* configurations set */
                        $$              = calloc(1, sizeof(
                            struct confnodes));

                        /* link the child nodes */
                        if ($1 != NULL)
142                             $1->parent = $$;
                        $2->parent      = $$;

                        $$->confs       = $1;
                        $$->conf        = $2;
147                 }
            ;

    configurations: configurations configuration
                {
152                     /* configurations set */
                        $$              = calloc(1, sizeof(
                            struct confnodes));

                        /* link the child nodes */
                        if ($1 != NULL)
157                             $1->parent = $$;
                        $2->parent      = $$;

                        $$->confs       = $1;
                        $$->conf        = $2;
162                 }
            |
                {
                        $$ = NULL;
                }
167         ;

    configuration: CONFIGURATION IDENTIFIER OPEN_BRACE newlines
        conf_param newlines conf_param newlines conf_param newlines
        conf_param newlines conf_param newlines conf_param newlines
        conf_param newlines CLOSE_BRACE newlines
                {
                        /* configuration node */
172                     $$              = calloc(1, sizeof(
                            struct confnode));

                        /* init */
```

```
                $2->type        = "configuration_id";
                $$->id          = $2;
177
                if (strcmp($2->name, "policy") == 0) {
                        policy_conf_id = conf_counter;
                }

182             $2->value       = conf_counter;
                $$->counter     = conf_counter;

                /* set ids */

187             if (strcmp($5->type, "keyword")) {
                        if ($5->conf_num == 0) {
                                $5->id = app_id_counter;
                                $5->conf = $$;
                                ++app_id_counter;
192                     }
                }

                if (strcmp($7->type, "keyword")) {
                        if ($7->conf_num == 0) {
197                             $7->id = net_id_counter;
                                $7->conf = $$;
                                ++net_id_counter;
                        }
                }
202
                if (strcmp($9->type, "keyword")) {
                        if ($9->conf_num == 0) {
                                $9->id = addr_id_counter
                                    ;
                                $9->conf = $$;
207                             ++addr_id_counter;
                        }
                }

                if (strcmp($11->type, "keyword")) {
212                     if ($11->conf_num == 0) {
                                $11->id = qoi_id_counter
                                    ;
                                $11->conf = $$;
                                ++qoi_id_counter;
                        }
217             }

                if (strcmp($13->type, "keyword")) {
                        if ($13->conf_num == 0) {
```

```
                                        $13->id = mac_id_counter
                                            ;
                                        $13->conf = $$;
                                        ++mac_id_counter;
                                }
                        }

                        if (strcmp($15->type, "keyword")) {
                                if ($15->conf_num == 0) {
                                        $15->id =
                                            addr_id_counter;
                                        $15->conf = $$;
                                        ++addr_id_counter;
                                }
                        }

                        if (strcmp($17->type, "keyword")) {
                                if ($17->conf_num == 0) {
                                        $17->id =
                                            radio_id_counter;
                                        $17->conf = $$;
                                        ++radio_id_counter;
                                }
                        }

                        /* link child nodes */
                        $$->app         = $5;
                        $$->net         = $7;
                        $$->net_addr    = $9;
                        $$->qoi         = $11;
                        $$->mac         = $13;
                        $$->mac_addr    = $15;
                        $$->radio       = $17;

                        ++conf_counter;
                }
        ;

conf_param: MODULES
                {
                        $$ = $1;

                }
        |       NOTHING
                {
                        $$ = $1;
                }
        ;
```

```
267
defined_events: defined_events event_condition
                {
                        /* event-conditions set */
                        $$              = calloc(1, sizeof(
                            struct eventnodes));
272
                        /* link child nodes */
                        if ($1 != NULL) {
                                $1->parent = $$;
                                last_evens = $$;
277                     }
                        $2->parent = $$;

                        $$->evens       = $1;
                        $$->even        = $2;
282             }
        |
                {
                        $$ = NULL;
                }
287     ;

event_condition: EVENT_CONDITION IDENTIFIER OPEN_BRACE
    IDENTIFIER RELOP CONSTANT CLOSE_BRACE newlines
                {
                        /* event-condition node */
292                     $2->value       = event_counter;
                        $2->type        = "event_id";
                        $2->lib         = $4->lib;

                        /* init */
297                     $$ = calloc(1, sizeof(struct eventnode))
                            ;
                        $$->id          = $2;
                        $$->counter     = $2->value;

                        /* link child nodes */
302                     $$->src         = $4;
                        $$->cst         = $6;
                        $$->cst->value  = editConst($6);

                        struct evtab *ev= evlook($2->name);
307                     ev->num         = event_counter;
                        ev->op          = $5;
                        ev->value       = $$->cst->value;

                        event_counter++;
312             }
```

```
        ;

policies: policies policy
                {
317                     /* policies set */
                        $$ = calloc(1, sizeof(struct policies));

                        /* link child nodes */
                        if ($1 != NULL)
322                             $1->parent = $$;
                        $2->parent = $$;

                        $$->pols        = $1;
                        $$->pol         = $2;
327             }
        |
                {
                        $$ = NULL;
                }
332     ;

policy: FROM from_configurations GOTO IDENTIFIER WHEN
    when_events newlines
                {
                        /* policy node */
337                     $$              = calloc(1, sizeof(
                            struct policy));

                        /* link child nodes */
                        $$->from        = $2;
                        $$->to          = $4;
342
                        $$->mask_l      = 0;
                        $$->mask_r      = -1;
                        $$->mask_l      = $6;
                        $$->counter     = policy_counter;
347
                        ++policy_counter;
                }

        |       FROM from_configurations GOTO IDENTIFIER WHEN
            when_events OR when_events newlines
352             {
                        /* policy node */
                        $$              = calloc(1, sizeof(
                            struct policy));

                        /* link child nodes */
357                     $$->from        = $2;
```

```
                        $$->to          = $4;

                        $$->mask_l      = 0;
                        $$->mask_r      = 0;
                        $$->mask_l      = $6;
                        $$->mask_r      = $8;
                        $$->counter     = policy_counter;

                        ++policy_counter;
                        ++policy_counter;
                }

        ;

from_configurations: IDENTIFIER
                {
                        $$ = $1;
                }

        |       ANY
                {
                        $$ = $1;
                }
        ;

when_events: one_event AND when_events
                {
                        $$ = $1 + $3;
                }
        |       one_event
                {
                        $$ = $1;
                }
        ;


one_event: IDENTIFIER
                {
                        /* iterator */
                        struct evtab *ep = NULL;
                        /* flag */
                        int found = 0;

                        /* check for undeclared identifiers */
                        for (ep = evtab; ep < &evtab[NEVS]; ep
                            ++)
                                if (ep->name && !strcmp(ep->name
                                    , $1->name)) {
                                        /* found */
```

```
                                        found = 1;
                                        break;
407                             }

                        /* undeclared event identifier */
                        if (!found)
                                yyerror("undeclared event-
                                    condition identifier");
412
                        $$ = 1 << (($1->value) - 1);
                }
        | NOT IDENTIFIER
                {
417                     /* iterator */
                        struct evtab *ep = NULL;
                        /* flag */
                        int found = 0;

422                     /* check for undeclared identifiers */
                        for (ep = evtab; ep < &evtab[NEVS]; ep
                            ++)
                                if (ep->name && !strcmp(ep->name
                                    , $2->name)) {
                                        /* found */
                                        found = 1;
427                                     break;
                                }

                        /* undeclared event identifier */
                        if (!found)
432                             yyerror("undeclared identifier")
                                    ;

                        /* create new event in symtab */
                        int len = strlen($2->name) + strlen("
                            not_") + 1;
                        char *new = calloc(len, 1);
437                     (void)snprintf(new, len, "not_%s", $2->
                            name);
                        struct symtab *sp = symlook(new);

                        if (!sp->type) {
                                sp->value = event_counter;
442                             sp->type = strdup($2->type);
                                sp->lib = $2->lib;

                                /*
                                 * make a new event entry by
                                     negating the
```

```
447                                     * operator
                                        */
                                       struct evtab *ev_old = evlook($2
                                           ->name);
                                       struct evtab *ev_new = evlook(sp
                                           ->name);

452                                    ev_new->num = event_counter;
                                       ev_new->op = negateOperator(
                                           ev_old->op);
                                       ev_new->value = ev_old->value;

                                       event_counter++;
457                            }

                               $$ = 1 << ((sp->value) - 1);
                       }
               ;
462

initial_configuration: START IDENTIFIER newlines
                       {
                               /* start node */
467                            $$ = calloc(1, sizeof(struct initnode));

                               /* link child nodes */
                               $$->id = $2;
                               $$->init = symlook($2->name)->value;
472                    }
               ;


virtual_networks: virtual_networks virtual_network
477                    {
                               $$              = $1;
                       }
               |
                       {
482                            $$ = 0;
                       }
               ;

virtual_network: VIRTUAL_NETWORK IDENTIFIER OPEN_BRACE
    configuration_ids CLOSE_BRACE newlines
487                    {
                               $$              = 0;
                               $2->type        = "virtual_id";
                               $2->value       = virtual_counter;
```

```
492
                        ++virtual_counter;
                }
        ;

497 configuration_ids: configuration_ids IDENTIFIER
                {
                        $$ = $1;
                }

502     |
                {
                        $$ = NULL;
                }
        ;
507
/* Library section */


library: newlines definitions
512     ;

definitions: definitions definition
        |
        ;
517
definition: USE type IDENTIFIER PATH
                {
                        /* iterator */
                        char *p = NULL;
522
                        /* lookup */
                        struct symtab *sp = NULL;

                        /* check for library re-declarations */
527                     if ((sp = symlook($3->name)) != NULL &&
                                                sp->type != NULL
                                                    )
                                /* failed */
                                yyerror("redeclaration of
                                    library");

532                     /* fix the child properties */
                        $3->type = $2;
                        $3->lib = $4;

                        /* differentiate based on the definition
                             type */
537                     if (!strcmp($3->type, "application")) {
```

```
                /* application */
                $4->type = TYPE_APPLICATION;
                $4->used = 0;
                $4->id = 0;
        }
        if (!strcmp($3->type, "network")) {
                /* network */
                $4->type = TYPE_NETWORK;
                $4->used = 0;
                $4->id = 0;
        }
        if (!strcmp($3->type, "qoi")) {
                /* qoi */
                $4->type = TYPE_QOI;
                $4->used = 0;
                $4->id = 0;
        }
        if (!strcmp($3->type, "mac")) {
                /* mac */
                $4->type = TYPE_MAC;
                $4->used = 0;
                $4->id = 0;
        }
        if (!strcmp($3->type, "radio")) {
                /* radio */
                $4->type = TYPE_RADIO;
                $4->used = 0;
                $4->id = 0;
        }
        if (!strcmp($3->type, "address")) {
                /* address */
                $4->type = TYPE_ADDRESS;
                $4->used = 0;
                $4->id = 0;
        }
        if (!strcmp($3->type, "source")) {
                /* source */
                $4->type = TYPE_EVENT;
                event_id_counter++;
                $4->used = 0;
                $4->id = event_id_counter;
        }

        /* extract the name from the path */
        if ((p = rindex($4->path, '/')) == NULL)
                $4->name = $4->path;
        else
                $4->name = ++p;
```

```
587                             $4->used = 0;

                                /* save the name as it was defined */
                                $4->def = $3->name;

592                     }
                        newlines
                ;

type: APPLICATION { $$ = "application"; }
597             | NETWORK { $$ = "network"; }
                | QOI { $$ = "qoi"; }
                | MAC { $$ = "mac"; }
                | RADIO { $$ = "radio"; }
                | ADDRESS { $$ = "address"; }
602             | SOURCE { $$ = "source"; }
                ;

newlines: newlines newline
                |
607             ;

newline: LF
                ;

612 %%

char program_file[PATH_SZ];
char library_file[PATH_SZ];
int done = 0;
617
extern FILE *yyin;
extern char *yytext;
extern int yyleng;

622 /* main entry point */
int
start_parser(int argc, char *argv[]) {

                /* check the file extention */
627             if (rindex(argv[1], '.') != NULL && !strcmp(rindex(argv
                    [1], '.'), ".sfp")) {
                        argv[1][strlen(argv[1])-4] = '\0';
                }

                /* init */
632             (void)memset(program_file, 0, PATH_SZ);
                (void)memset(library_file, 0, PATH_SZ);
```

```
        (void)snprintf(library_file, PATH_SZ, "%s.sfl", argv[1])
            ;
        (void)snprintf(program_file, PATH_SZ, "%s.sfp", argv[1])
            ;

637     /* process libraries */
        lineno = 1;
        tokenpos = 0;

        /* cleanup */
642     (void)atexit(gc);

        /* open the specific library file */
        yyin = fopen(library_file, "r");

647     /* try fennec fox standart libray located at (
            $FENNEC_FOX_LIB)/STD_FENNEC_FOX_LIB */
        if (!yyin) {
                (void)snprintf(library_file, PATH_SZ, "%s/%s",
                    getenv("FENNEC_FOX_LIB"), STD_FENNEC_FOX_LIB)
                    ;
                yyin = fopen(library_file, "r");
        }
652
        if (!yyin) {
                /* failed */
                (void)fprintf(stderr,
                        "%s.sfl: no such file or directory and
                            no standard library\n",
657                     argv[1]);
                exit(1);
        }

        /* main loop */
662     do {
                initialize();
                yyparse();

        } while(!feof(yyin));
667
        /* byeZzz */
        return 0;
}

672 /* error reporting */
yyerror(char *errmsg) {

        /* error in program */
        if (done)
```

```
677                 (void)fprintf(stderr, "\nsfc in program: %s\n",
                        program_file);
        else
                (void)fprintf(stderr, "\nsfc in library: %s\n",
                        library_file);

        /* line */
682     (void)fprintf(stderr, "%s at line %d, position %d:\n",
            errmsg, lineno,
                                                        tokenpos
                                                            -
                                                        yyleng
                                                            + 1)
                                                        ;
        (void)fprintf(stderr, "%s\n", linebuf);
        (void)fprintf(stderr, "%*s\n", tokenpos - yyleng + 1, "^
            ");

687     /* terminate */
        exit(1);
}

/* restart the parser with the program file */
692 int
yywrap(void) {
        /* finish; done with library and program */
        if (done)
                return 1;
697     else {
                /* re-init */
                lineno          = 1;
                tokenpos        = 0;

702             /* open the program file */
                yyin            = fopen(program_file, "r");
                if (!yyin) {
                        /* failed */
                        (void)fprintf(stderr,
707                             "%s: no such file or directory\n
                                    ",
                                program_file);
                        exit(1);
                }

712             /* done with the library */
                done = 1;
        }

        /* default */
```

```
717             return 0;
    }

    /* symbol lookup */
    struct symtab *
722 symlook(char *s) {
            /* iterator */
            struct symtab *sp = NULL;

            /* loop */
727         for(sp = symtab; sp < &symtab[NSYMS]; sp++) {
                    /* is it already here? */
                    if (sp->name && !strcmp(sp->name, s))
                            return sp;

732                 /* is it free */
                    if(!sp->name) {
                            sp->name = strdup(s);
                            return sp;
                    }
737                 /* otherwise continue to next */
            }
            yyerror("symtab is full");
    }

742
    /* symbol */
    struct symtab *
    find_sym(char *s) {
            /* iterator */
747         struct symtab *sp = NULL;

            /* loop */
            for(sp = symtab; sp < &symtab[NSYMS]; sp++) {
                    /* is it already here? */
752                 if (sp->name && !strcmp(sp->name, s))
                            return sp;

            }
            return NULL;
757 }

    /* process_module */
    struct modtab *
    proc_module(char *s) {
762         /* save params */
            char *params = strdup(rindex(s, '('));

            /* symtab ptr */
```

```
        struct symtab *sp = find_sym(strtok(s, "("));
767
        /* iterator */
        struct modtab *mp = NULL;

        /* check if this is a know module */
772     if (sp == NULL)
                yyerror("no such module");

        /* loop */
        for(mp = modtab; mp < &modtab[NSYMS]; mp++) {
777             /* is it free */
                if(!mp->name) {
                        mp->name = strdup(sp->name);
                        mp->id = 0;
                        mp->type = strdup(sp->type);
782                     mp->lib = sp->lib;
                        mp->params = strdup(params);
                        mp->conf = NULL;
                        mp->conf_num = 0;
                        return mp;
787             }
                if (strcmp(mp->name, sp->name) == 0 &&
                        strcmp(mp->params, params) == 0) {
                        return mp;
                }
792     }
        yyerror("modtab is full");
}

/* library lookup */
797 struct libtab*
liblook(char *l) {

        /* iterator */
        struct libtab *lp = NULL;
802
        /* loop */
        for(lp = libtab; lp < &libtab[NLIBS]; lp++) {
                /* is it already here? */
                if (lp->path && !strcmp(lp->path, l))
807                     yyerror("library already exists");

                /* is it free */
                if(!lp->path) {
                        lp->path = strdup(l);
812                     checkForRemotePath(lp);
                        lp->used = 0;
                        return lp;
```

```
                }

817             /* otherwise continue to next */
        }
        yyerror("libtab is full");
}

822 /* event-condition lookup */
struct evtab*
evlook(char *name) {

        /* iterator */
827     struct evtab *ev = NULL;

        /* loop */
        for(ev = evtab; ev < &evtab[NEVS]; ev++) {
                /* found */
832             if (ev->name && !strcmp(ev->name, name))
                        return ev;

                /* insert */
                if (!ev->name) {
837                     ev->name = strdup(name);
                        return ev;
                }
        }

842     /* failed */
        yyerror("evtab is full");
}

/* initialize the keywords set */
847 void
initialize(void) {

        /* keywords set */
        char *keywords[] = {"configuration", "start", "use", "
            application",
852                     "network", "source", "event-condition",
                            "from", "goto",
                        "none", "conf", "event", "on", "off",
                        "qoi", "mac", "radio", "virtual-network"
                            ,   // new keywords
                        "when", "nothing", "any", "once", "event
                            "};

857     /* size of the keywords set */
        int k_num               = sizeof(keywords)/sizeof(char*)
            ;
```

```
        /* iterate */
        int i                   = 0;
        struct symtab *sp       = NULL;

        /* init */
        for(i = 0, sp = symtab; i < k_num; i++, sp++) {
                sp->name = keywords[i];
                sp->value = 0;
                sp->type = "keyword";
        }
}

/* get the negation of an operator */
int
negateOperator(int i) {
        switch(i) {
                case LT:
                        return GE;
                case GT:
                        return LE;
                case LE:
                        return GT;
                case GE:
                        return LT;
                case NE:
                        return EQ;
                case EQ:
                        return NE;
                default:
                        yyerror("unknown RELOP operator");
        }
}

/* parse constants */
int
editConst(struct symtab *entry ) {

        /* get the constant */
        char *sp = entry->name;

        /* extract the integer part */
        int v = atoi(sp);
        for (; isdigit(*sp); sp++);

        /* set the type accordingly */

        /* sec */
        if (!strcmp(sp, "sec")) {
```

```
                entry->type = strdup("timer");
                v *= SEC_CONV;
                return v;
        }

        /* min */
        if (!strcmp(sp, "min")) {
                entry->type = strdup("timer");
                v *= MIN_CONV;
                return v;
        }

        /* hr */
        if (!strcmp(sp, "hr")) {
                entry->type = strdup("timer");
                v = HR_CONV;
                return v;
        }

        /* Celsius */
        if (!strcmp(sp, "C")) {
                entry->type = strdup("temperature");
                return v;
        }

        /* Fahrenheit */
        if (!strcmp(sp, "F")) {
                entry->type = strdup("temperature");
                v = (v - 32) * 5 / 9 ;
                return v;
        }

        /* default */
        entry->type = strdup("number");

        return v;
}

printTable() {

       struct symtab *sp;
        printf("\n");
        for(sp = symtab; sp < &symtab[NSYMS]; sp++) {
                /* is it already here? */
                if (sp->name ) {
                                printf("%s    %s    %d\n", sp->
                                    name, sp->type, sp->value);
                } else {
                        printf("\n\n");
```

```
                        break;
957                 }
            }

            struct evtab *ev;
            for(ev = evtab; ev < &evtab[NEVS]; ev++) {
962                 if (ev->name) {
                        printf("%s      %d      %d      %d\n", ev->
                            name, ev->num, ev->op, ev->value);
                    } else {
                        printf("\n\n");
                        return;
967                 }
            }
}

/* garbage collection */
972 void
gc(void) {

            /* cleanup */
            (void)close(yyin);
977 }

/* check if the path is remote */
void
checkForRemotePath(struct libtab *lp) {
982
            /* check if it is http:// */
            if (!strncmp(lp->path, "http://",7)) {

                    /* make temp dir for all downloaded libraries */
987                 mkdir(TEMP_DIR, S_IRWXU);

                    char *p = NULL;
                    /* extract the name from the path */
                    if ((p = rindex(lp->path, '/')) != NULL) {
992                         p++;
                    }

                    /* make dir for this library */
                    char *new = malloc(strlen(p)+strlen(TEMP_DIR)+2)
                        ;
997                 sprintf(new, "%s/%s", TEMP_DIR, p);
                    mkdir(new, S_IRWXU);

                    /* prepare system command */
                    char *command = malloc(strlen(lp->path)+strlen(
                        new)+40);
```

```
1002                    sprintf(command, "wget -q -nd -r --no-parent -A.
                            nc -P %s %s", new, lp->path);

                        if( system(command) ) {
                                fprintf(stderr, "Swift Fox Error: Could
                                    not download files from %s\n", lp->
                                    path);
                                exit(1);
1007                    }
                        free(lp->path);
                        lp->path = strdup(new);
                        free(command);
                        free(new);
1012            }
        }
```

../../src/sf/sf.y

References

[1] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, & Tools*. Addison-Wesley, 2nd edition, 2007.

[2] Luca Cardelli. Typeful programming. *Formal Description of Programming Concepts*, pages 431–507, 1991.

[3] Stephen C. Johnson. Yacc: Yet another compiler-compiler. Technical Report 32, July 1975. Computing Science Technical Report, Bell Laboratories.

[4] Michael E. Lesk and Eric Schmidt. Lex: A lexical analyzer generator. *UNIX Research System: Programmer's Manual (10th edition)*, 2:375–387, 1990.

[5] Bison-GNU parser generator. Gnu. Online, March 2010.

[6] Vern Paxson, Jef Poskanzer, and Kevin Gong. Flex–fast lexical analyzer generator. Online, March 2010.

[7] Marcin Szczodrak, Vasileios P. Kemerlis, Xuan Linh Vu, and Yiwei Gu. Swift fox tutorial. Technical report, March 2010. Computer Science Department, Columbia University.

[8] Marcin Szczodrak, Vasileios P. Kemerlis, Xuan Linh Vu, and Yiwei Gu. Swift fox whitepaper. Technical report, February 2010. Computer Science Department, Columbia University.