FIREDRL

Language Final Report

COMS 4115
Programming Languages and Translators

Marvin J. Rich
e-mail: mjrich@us.ibm.com

1.0 Introduction

The internet is a ubiquitous entity that touches all aspects of modern society. The services provided by the underlying protocols have enhanced business and personal communication to a worldwide scope. As the internet has integrated into the normal processes conducted by society, security has emerged as a necessary component for providing services that can be accessed worldwide. For example, a Local Area Network (LAN) configuration representing a business typically has data online that must be protected from unauthorized access, as well as data that must be accessible by everyone, in order to conduct a competitive business. These opposing requirements must co-exist on a common set of applications and protocols, which makes security of the resulting LAN configuration a complex undertaking for network administrators.

Firewalls are a first line of defense for LAN security. They are not the only component of network security, but an important one nonetheless. A firewall imposes and enforces rules on the type of traffic and services allowed to flow in or out of a network. Typically a firewall will be implemented at a router, which is an entrance/exit point for a LAN. In order to implement the various levels of security and services for a typical LAN, several routers are usually present, which partition the total LAN into segments with varying degrees of access. Firewalls will generally be implemented on each of these routers to enforce a degree of access rights, but may be implemented on host machines also. For large LANs, it is difficult to determine the security imposed by various levels of firewalls, such that the intended services controlled by the firwalls are indeed provided at each host on the LAN.

The Firewall Integrity Review Exploit Description Report Language, FIREDRL (pronounced “fire drill”), is a rapid LAN configuration and exploitation test language. This language is specific to the domain of firewalls. It allows a network administrator, to design the firewall aspects of a network by defining the network, testing fundamental exploits, and reporting the affects of the attack on hosts of the network. This process can be repeated until the administrator is satisfied with the operation of the firewalls in the LAN. The FIREDRL language provides the following capabilities:

- Easy LAN Prototyping
- Easy Firewall rule specification
- Rapid Firewall Exploit Testing
- Flexible Exploit Result Reporting

1.1 Overview

For explaining the capabilities of the FIREDRL language, a typical LAN configuration is depicted in Figure 1. The LAN contains 3 segments labeled X,Y and Z respectively. LAN segment X has two routers (R1,R2), and 3 hosts (A,B,C). LAN Segment Y has two routers (R2,R3) and also 3 hosts (D,E,F). LAN segment Z has 1 router (R3) and 2 hosts (G,H). Each LAN segment typically serves some functional requirement. For this illustration, the segments denote different security policies as to what can be accessed. The firewalls are implemented on all the routers for this illustration (though a firewall can also be implemented on a host machine).

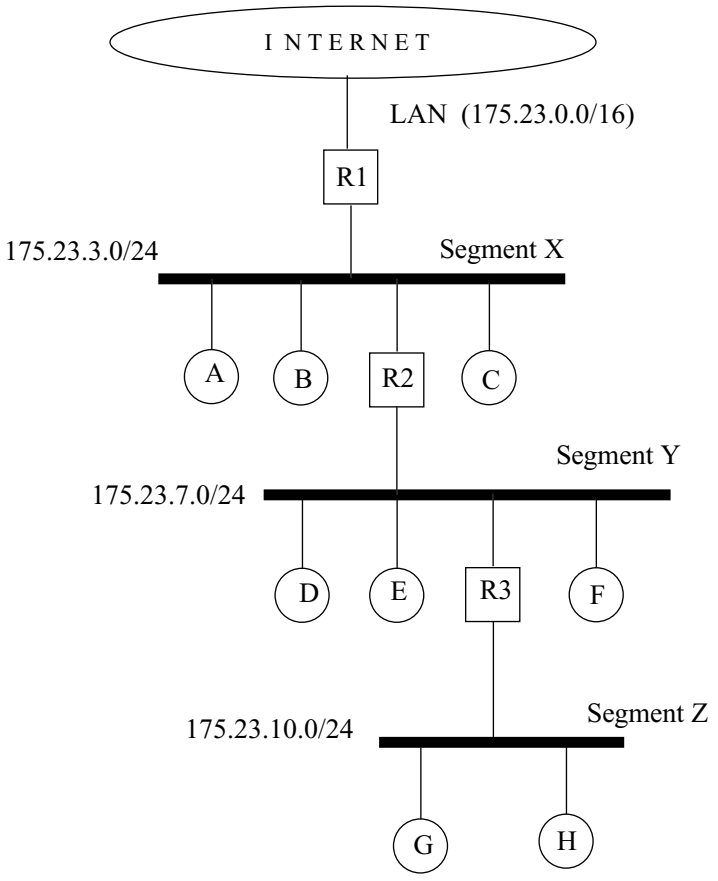

Figure 1: Typical Multi-Segment LAN Configuration

Rapid LAN and Firewall Configuration

The FIREDRL language can easily lay out the topology of a LAN, which is the environment that an exploit attack is analyzed against. Using the example in figure 1, the FIREDRL statements required to describe the LAN are as follows:

```
1   //Define total lan segment topology
2   MyLan       = Segments{Segment_X,Segment_Y,Segment_Z};
3   Segment_X = hostname{A,B,R1,R2,C};
4   Segment_Y = hostname{D,E,R2,R3,F};
5   Segment_Z = hostname{G,R3,H};
6
7   //Define host_ids in each segment
8   Segment_X = hostid{10,20,30,40,50};
9   Segment_Y = hostid{2,3,41,25,61,};
10  Segment_Z = hostid{34,35};
11
12  //Define IP Addresses and Subnets of LAN
13  MyLan       = ip(175.23.0.0/16);
14  Segment_X = ip(175.23.3.0/24);
15  Segment_Y = ip(175.23.7.0/24);
16  Segment_Z = ip(175.23.10.0/24);

17  //Filter rules
18  Segment_X.R1 = filter(IN,PORT=80, YES);
19  Segment_Y.R2 = filter(IN,PORT=25,NO);
```

Lines (2-5) establish the overall network topology by assigning names to key network hosts and routers a top-down fashion. FIREDRL can deduce the firewall hosts by nature that the host name will exist in multiple network segments. Lines (7-10) assigns unique (semantic check) host IDs to the established hosts. Finally lines 13-16 assign IP addresses to the topology. Subnet mask information is appended to the IP address. For example a "/24" string appended to an IP address indicates that the first 24 bits are the network address, which implies a 255.255.255.0 subnet mask. Lines 18-19 configure firewall rules on routers R1 in Segment_X and R2 in Segment_Y respectively. One can filter incoming or outgoing traffic based on the rules. Of course the language is free format with line comments allowed (lines 1,7,12 and 17).

Rapid Exploit Testing and Reporting

An exploit attack is a specific method used to attempt to penetrate the LAN from the internet. FIREDRL provides control instructions which allow logical decisions that can be made to guide an attack sequence. Common exploits are packaged with the support such that only control decisions need be mapped out in the test. An example in FIREDRL is as follows:

```
1   //Issue ICMP TTL Probe Attack
2   success = 0;
3   for (TTL = 2 to 20 by 1) {
4      ICMP(echo,TTL);
5      if  (resp.success)
6         print("Successful Probe for TTL=", TTL);
7   }
```

This attack uses ICMP echo requests, with increasing TTL value, to discover the routing depth in the LAN. This information provides topology clues to an attacker. Familiar programming control constructs such as "for" and "if" statements in lines 3 & 5 allow decisions to be made while progressing through an attack. The control flow constructs are wrapped around an attack directive (line 4) to realize a wide ranging probe of the network. Print statements, as depicted in line 6 provide a method for displaying the results of the attack on the network. The user is free to print the status of interest, based on the placement of print statements.

1.2 Implementation Model

The underlying execution language is Java. The FIRDRL language is compiled into Java executables that, given a network topology and an exploit definition, generate a report on how the exploit was handled by the firewalls depicted in the network. The actual internet exchanges are abstracted to object message/response transactions between the attacker and the network. The attacker serves as the client, probing messages to gain knowledge about the network. The network description serves as the server, which provides responses based on the filtering rules specified in its firewall setup rules (or lack thereof).

1.3 Availability

The Beta version 1.0 is scheduled for release in December of 2005. A language manual will preceed the delivery in December, which will describe the version 1.0 capabilities. The targeted capabilities for the version 1.0 delivery are as follows:

- Full LAN specification support
- Firewall Filter Rules Support
- Exploit control language support
- Three exploits (Ping Discovery, ICMP TTL Probe, and Port Scan)
- Exploit Results Reporting

1.4 Conclusion: The FIREDRL Advantage

The FIREDRL language allows for rapid LAN prototyping and firewall testing. Rapid LAN prototyping is realized by abstracting the LAN down to its key hardware and software components, as viewed from a firewall. The key hardware components are hosts and routers. The key software components are firewall rule sets, IP addresses, and TCP ports. Rapid firewall testing involves a simple control language wrapped around a set of exploitation attack abstractions. This allows easy testing of an exploit as an attacker would invoke it, by using the control constructs to easily direct an exploit to a range of targets.

The FIREDRL advantage is that the abstract prototyping constructs incorporated in the language affords the user a mechanism to gain in depth knowledge of the firewall capabilities of his/her network without a large expenditure in actual network design and testcase effort. The abstraction level also provides a steep performance advantage. The firewall capabilities are summarized for the entire LAN, such that knowledge of a LAN can be gained in a shorter period of time. This ultimately results in a faster implementation of a total LAN security solution through rapid stabilization of the firewall security component.

2.0 Tutorial

The Firedrl language facilitates rapid prototyping of network configurations for the purpose of firewall analysis. The language is utilized to specify the network components pertinent to the exchange of information between other networks, as well as the components responsible for controlling access to hosts on a configured test network. The network components consists of hosts, LANs and networks. A host represents a single computer which can send or receive packets of information. A LAN (short for Local Area Network) is a collection of hosts with a common network address. A network is a collection of LANs which represent a particular configuration of interest. A firewall controls the exchange of information by imposing rules on what information may enter or leave a host. Typically a particular host, called a router, is responsible for controlling information entering or leaving a LAN, by providing at least a one other network address for information exchange with another network or LAN. The Firedrl language abstracts a firewall as a set of rules that control what information can enter or leave a particular host. With the ability to define the network and rules for information exchange, one can send packets of information to a test network to determine if access protection to individual hosts meets requirements. This tutorial describes how to utilize the Firedrl language to prototype and test a network firewall configuration.

2.1 Network Specification

The first phase of utilizing the Firedrl language is to define your test network and the rules governing information exchange on that network. The Firedrl language defines declaration statements for specifying hosts, LANs which represent the fundamental components of the language. LANs are identified functionally their network address, such that a LAN declaration also provides this identification (see LRM for details). The test network consists of the set of defined LANs.

Example: Define four hosts named “A” , “B”, “C”, and “D” respectively:

```
host_def A,B,C,D;
```

Example: Define two LANs named “lan1” and “lan2” respectively:

```
lan_def Lan1:10.2.0.0/16, Lan2:10.2.4.0/8;
```

2.1 Rule Specification

The access controls are provided in the rule definitions. Each rule has a name for referencing by multiple hosts . A rule defines an action to be applied to a packet (block or allow passage) flowing in a particular direction (in, out or both). The rest of the rule filters on the type of packet (TCP,

UDP, ICMP), source or destination address, source or destination port (where applicable), and other control fields.

Example: Define Rule1 to block TCP input to port 80 from any source to any destination:

```
rule Rule1 = block in * *  TCP  * 80  0;
```

2.2 Component Connection

Once the hardware components and rules are defined, they can be connected to formulate the communication paths and access controls. Hosts are connected to LANs and rules are connected to hosts. The Firedrl language implicitly connects LANs to LANs when a host name is shared by two different LANs. Rules are shared in a similar manner: when they are referenced by more than one host. A host is also assigned a LAN id when connected to a LAN.

Example: Connect host A and B to Lan1 and host B and C to Lan2. Host A and C will have ids 12 and 24 respectively on Lan1, and hosts B and C will have ids 18 and 21 respectively on Lan2. Note that host B has implicitly become a router, since it has an interface on Lan1 and Lan2:

```
Lan1 @=  A:12, B:24;
Lan2  @=   B:18, C:21:
```

Example: Connect Rule1 to host A and Rule2 and Rule3 to host B:

```
A  %=  Rule1;
B   %=   Rule2, Rule3;
```

2.2 Driver Specification

Once the hardware components and rules are defined, and connected, the test Firedrl components are then utilized to send packet information and observe results. The Firedrl probe statement is used to stimulate the network configuration with packet data. The report statement is used to control which hosts respond with report data. The intended report statement, which sets up who will report status should be specified before the intended probe statement. Typically control statements (loop and if statements) are framed around these statements to drive multiple tests.

Example: Report TCP access coming in to any host:

```
report target= * protocol = TCP  direction = send;   %=  Rule1;
```

Example: Send a TCP packet to test network.

```
snd_tcp(src=10.74.5.4#2 dest=1.24.2.1#80 cntl= 0;
```

2.2 Compiling and Execution

One compiles the FIredrl Lexer,Parser, and Walker into same directory as the Firedrl class. To invoke the language enter:

```
java Firedrl  <src_file>
```

The src_file is the file that contains the FIredrl language statements. The output reports are written to System.out, and therefore can be redirected to a file.

3.0 Language Reference Manual

The purpose of this manual is to convey the syntax and semantics of the FIREDRL language (*F*irewall *I*ntegrity *R*eview *E*xploit *D*etection *R*eport *L*anguage). Syntax and examples of valid instances of each statement are provided. A BNF format grammar utilize to specify each statement syntax in a concise manner. The semantics associated with each statement is also provided.

3.1 Lexical Conventions

Each statement in the Firedrl language consists of a set of fundamental tokens, where the sequence of tokens uniquely identify a valid statement. This section defines the set of tokens and keywords used to formulate valid Firedrl language statements during lexical analysis. Lexical tokens will be capitalized in order to easily distinguish them in the BNF grammar for language statements.

3.1.1 Separators:

Blanks (whitespace), tabs, new line, and carriage return are ignored when the language is scanned during lexical analysis. These separators are used to separate tokens, and to provide a measure of flexibility in the input format for readability (can have an arbitrary number of consecutive separator characters between tokens).

BNF Grammar:

WS : (blank | tab | newline | carage_return | newline carage-return)
blank : ‘ ‘
tab : ‘\t’
newline : ‘\n’
carage_return : ‘\r’

Other single character separators also exist as follows:
SEMI : ‘ ‘
COMMA : ‘\t’
COLON : ‘\n’
LPAREN : ‘(‘
RPAREN : ‘)’
LBRKT : ‘{‘
RBRKT : ‘}’
SLASH : ‘/’
POUND : ‘#’
STAR : ‘*’
DOT : ‘.’

3.1.2 Constants:

Integers are the only numeric constants defined in the Firedrl language. An integer constant is defined as a string of one or more decimal digits from ‘0’ to ‘9’:

BNF Grammar:

INT : (DIGIT)+
DIGIT : ‘0’ ... ‘9’

3.1.3 Identifiers:

Indentifiers are used to label network objects and local variables in Firedrl. A valid identifier must consist of an initial alphabetic character, followed by zero or more alphabetic, decimal, or ‘_’ characters.

BNF Grammar:

*ID: ALPHA (ALPHA | DIGIT | ‘_’)**
ALPHA : (‘a’ .. ‘z’ | ‘A’..’Z’)

Examples:

This_is_valid
atemp_name
A
B1_B2

3.1.4. Operators:

Operators indicate the transformation or comparison of objects or data.. The ‘:=’ operator is used to define host and network objects in Firedrl’s network domain. The ‘@=’ operator is used to assign hosts to a network. The ‘<-’ operator is used to assign packet filtering rules to a host. Regular variable assignments utilize the ‘=’ operator. The standard set of comparison operators are also defined (‘>’,’<‘,’==’, ‘<=’, ‘>=’) . The iteration operator ‘+=’ supports aggregation of host and network attributes within a loop.

Grammar:

DEFINE : ‘:=’
EQ : ‘=’
H_ASSIGN : ‘@=’
R_ASSIGN : ‘<-’
VAR_ASSIGN : ‘=’
CMPR : (‘<‘ | ‘>’ | ‘==’ | ‘<=’ | ‘>=’)
INCR_ASSIGN : ‘+=’

3.1..5 Comments:

Line comments are ignored in Firedrl (treated like white space). A line comment starts with a "//" two character sequence. Upon encountering this character sequence, the remaining line text is ignored, including the initial '//' token. The remaining line text is defined to be any characters up to and including the end of line character(s). Therefore if a comment shares a line with a language statement, it should follow the statement.

Grammar:

comment : *com_token restline*
com_token : '//'
restline : (~('\n' | '\r'))* ('\n' | '\r')

Example:

```
//This is a standalone comment
<language stmt>     // This comment shares line with a statement
```

3.1.6 Keywords:

The following keywords are reserved, and should not be used as an identifier name:

do for if host network attack

3.2 Syntax Notation

A Firedrl language consists of one or more statements. Each statement can be in one of eight possible categories. This section will define the syntax of the statement or statements supported in each of the eight statement categories.

Grammar:

lang : (*stmt*)+
stmt : *host* | *network* | *rule* | *loop*
decision | *assignment* | *probe* | *report*

3.2.1 Host Declaration Statement:

A host statement defines one or more hosts that reside in the network configuration. Each host is defined by an identifier and at least one associated host interface address. If a host has more than one host interface address, then each host address must be qualified by the network name it is associated with. A host address consists of a "dot.Int" notation, where one can specify from 1 to 4 blocks of 8 bits. Each 8 bit block is separated by a dot if more than 8 bits are specified. The host interface address only requires enough bits encoded to represent the address right justified in a full 32 bit notation. An implicit cardinality is associated with each hostname such that the language supports looping through hostnames. The cardinality is assigned in the order the hosts are defined.

BNF Grammar:

host : "host_def" *DEFINE hostname (COMMA hostname)* SEMI*
hostname : *(ID COLON intfc | ID COLON intfc SLASH ID)*
intfc : INT (DOT INT (DOT INT (DOT INT)?)?)?

Examples:

```
host_def := HostA:45;
host_def := HostB:25, HostC:27;
```

The first example defines a single host, HostA, with a single interface address of decimal 45. The second example defines two hosts, HostB and HostC are assigned interface addresses 25 and 27 respectively.

3.2.2 Network Declaration Statement:

A network statement defines one or more networks that comprise the total network configuration. Each network is defined by an identifier and a associated 32 bit network address. The

network address consists of a 32 bit “quad-dot” notation followed by a slash and an integer number. The 32 bits are defined in four blocks of 8 bits, separated by dots, with a decimal encode of each of the 8 bits. The most significant bits designate the network address, and the least significant bits are set to zero. The integer value after the network address defines the number of bits, from most significant bit, that make up the network address. If a host has more than one host interface address, then each host address must be qualified by the network name it is associated with. A host address consists of a “dot.Int” notation, where one can specify up to 4 dotted integers (just need to define sufficient integers to represent the host address right justified in a full quad “dot.int” IP address). An implicit cardinality is associated with each network name such that the language supports looping through network names. The cardinality is assigned in the order the network names are defined.

BNF Grammar:

network : “lan_def” *DEFINE netwkname (COMMA netwkname)* SEMI*
netwkname : *ID COLON ipaddr SLASH INT*
ipaddr : INT DOT INT DOT INT DOT INT

Example:

```
lan_def :=  Internet:1.73.0.0/16,NetworkA:1.73.2.0/24;
```

This example defines two networks. The network called Internet has a network address of 1.73.x.x since the network is defined for the first 16 bits. The network called NetworkA has a network address of 1.73.2.x since the first 24 bits define the network address.

3.2.3 Rule Statements:

The functionality of each firewall is defined by rules that are attributed to the host interface. These rules must be defined before they can be assigned to a host interface. There are three

types of rules based on the target protocols of TCP, UDP and ICMP respectively. Each rule has fields specific to the protocol that designate what is to be allowed or blocked. A '*' in the source or destination address matches all IP addresses. The source and destination ports can be optionally specify logical comparison of port values. The connect field for TCP rules specifies which control bits are allowed to be set.

BNF Grammar:

rule : "rule" *ID 'DEFINE LPAREN*
(tcp_rule | udp_rule | icmp_rule)
RPAREN SEMI
tcp_rule : "TCP" *direction src_addr dest_addr src_port dest_port*
connect action
udp_rule : "UDP" *direction src_addr dest_addr src_port dest_port action*
icmp_rule : "ICMP" *direction src_addr dest_addr type action*
direction : ("in" | "out")
src_addr : "src=" (*ipaddr* | *STAR*)
dest_addr : "dest=" (*ipaddr* | *STAR*)
src_port : "src_port=" ((*CMPR*)? *INT* | *STAR*)
dest_port : "src_port=" ((*CMPR*)? *INT* | *STAR*)
connect : *INT*
type : *INT*
action : ("allow" | "block")

Examples:

```
rule Rule1 := TCP  in   *   1..74.2.45 *   >1024  block;
rule Rule3 := ICMP out  1.74.2.45   *   3   block;
```

Rule1 specifies that incoming TCP packets to ipaddr 1.74.2.45 with port greater than 1024 should be blocked. Rule2 specifies that ICMP type 3 responses from 1.74.2.45 should be blocked.

3.2.4 Loop Statements:

Loop statements allow for iterations. In addition to looping on a variable, a network and host loop is also defined. The network and host loops take advantage of the implicit cardinality assigned to network and host definitions, to loop through network or host names. A block of statements can be executed under each loop construct.

BNF Grammar:

loop : (*net_loop* | *host_loop* | *do_loop*)
net_loop : "for" "network" *LPAREN ID* ("to" *ID*)? *RPAREN*
LBRKT (*stmt*)* *RBRKT*
host_loop : "for" "host" *LPAREN ID* ("to" *ID*)? *RPAREN*
LBRKT (*stmt*)* *RBRKT*
do_loop : "do" *LPAREN ID EQ INT* ("to" *INT* ("by" *INT*)?)? *RPAREN*
LBRKT (*stmt*)* *RBRKT*

Examples:

```
for network(NetworkA to NetworkZ) { ..... }        //Loop through network names
for host(host1 to host10) {.....}                   //Loop through host names
do (var = 1 to 5) {.....}                          //Generic variable loop
```

3.2.5 Decision Statements:

Decision statements allow alternate segments of code to execute based on the outcome of a test predicate. Under a network loop context, a network if statement is defined. Under a host loop context, a host if statement is defined. The network and host if statement allows for unique code to execute, based on the network or host currently iterated on.

BNF Grammar:

decision : (*net_if* | *host_if* | *var_if*)
net_if : "if" *LPAREN* "network" *CMPR ID RPAREN*
LBRKT (*stmt*)* *RBRKT* ("else" *LBRKT* (stmt)* *RBRKT*)?
host_if : "if" *LPAREN* "host" *CMPR ID RPAREN*
LBRKT (*stmt*)* *RBRKT* ("else" *LBRKT* (stmt)* *RBRKT*)?
var_if : "if" *LPAREN* ID *CMPR INT RPAREN*
LBRKT (*stmt*)* *RBRKT* ("else" *LBRKT* (stmt)* *RBRKT*)?

Examples:

```
if (network == NetworkQ ) { ..... }        //if loop network name equals NetworkQ
if (host >= HostZ)   {.....}               //if loop host name >= HostZ
```

```
if (var > 5)  {.....}                    //Regular variable if statement
```

3.2.6 Assignment Statements:

Assignments are made to networks and hosts. Networks are assigned hosts and hosts are assigned rules. Rules actually apply to interfaces. Therefore if a host has more than one interface, the assignment should be made within a network loop (the associated interface to apply the rule will then be known). Increment assignments (e.g. +=) are also allowed. Types must be compatible in the assignments. A network must be assigned a hostname. A host must be assigned a rule name.

BNF Grammar:

assignment : (*net_assign* | *host_assign* | *incr_add*)
net_assign : *ID H_ASSIGN ID (PLUS ID)* SEMI*
host_assign : *ID R_ASSIGN ID (PLUS ID)* SEMI*
incr_add : *ID INCR_ASSIGN ID SEMI*

Examples:

```
Network1 @= hostB;                  // assign hostB to network Network1
hostB <-  Rule2;                    // assig Rule2 to hostB
for (Network1) { hostc <- Rule3; }  //hostc interface for Network1 assigned Rule3
Network1 += hostC;                  // add hostC to Network1
```

3.2.7 Probe Statements:

Probe statements initiate packet tests of the network configuration. They work in concert with variable control statements to send packets of various formats to illicit response from firewalls in the network. The three types of probe packets are TCP, UDP and ICMP packets respectively. Variables can be substituted for iteration ranges within loops.

BNF Grammar:

probe : (*tcp_pkt* | *udp_pkt* | *icmp_pkt*)
tcp_pkt : "snd_tcp" *LPAREN* "src=" *v_ipaddr POUND v_int*
"dest=" *v_ipaddr POUND v_int*
"cntl=" *v_int*
RPAREN SEMI
udp_pkt : "snd_udp" *LPAREN* "src=" *v_ipaddr POUND v_int*
"dest=" *v_ipaddr POUND v_int*
RPAREN SEMI
icmp_pkt : "snd_icmp" *LPAREN* "src=" *v_ipaddr*
"dest=" *v_ipaddr*
"type=" *v_int*
RPAREN SEMI
v_ipaddr : *v_int DOT v_int DOT v_int DOT v_int*
v_int : (*INT* | *ID*)

Examples:

```
snd_tcp(src=1.74.5.3#4756 dest=1.24.5.2#80 cntl=0);      //fixed tcp packet
snd_icmp(src=1.74.5.4 dest=1.75.2.1 type= 4);            //fixed icmp packet
snd_tcp(src=1.74.5.3#2072 dest=1.24.5.C#80 cntl=0);      //variable destination address
snd_icmp(src=1.74.5.4 dest=1.23.2.1 type=D).             //variable type field
```

3.2.8 Report Statement:

The report statement provides the capability to observe the affects of exploits attempted via probe statements. One can target reports to a specific host or obtain reports for all hosts. The report can be specific to a particular protocol, as well as a specific traffic direction. A special ID called "attack" is used to target reports from the initiator of packets to the network.

BNF Grammar:

report : "report" "target=" (*STAR* | "attack" | *ID*)
"protocol=" (*STAR* | "TCP" | "UDP" | "ICMP")
"direction=" (*STAR* | "snd" | "rcv")
SEMI

Examples:

```
report  target=* protocol=TCP  direction= *;          //report all TCP activity on all hosts
report  target=HostA protocol=ICMP direction=rcv;  //report ICMP pkts received on HostA
```

3.3 BNF Summary

lang : (*stmt*)+
stmt : *host* | *network* | *rule* | *loop*
decision | *assignment* | *probe* | *report*

host : "host_def" *DEFINE hostname (COMMA hostname)* SEMI*
hostname : *(ID COLON intfc | ID COLON intfc SLASH ID*)
intfc : INT (DOT INT (DOT INT (DOT INT)?)?)?

network : "net_def" *DEFINE netwkname (COMMA netwkname)* SEMI*
netwkname : *ID COLON ipaddr SLASH INT*
ipaddr : INT DOT INT DOT INT DOT INT

rule : "rule" *ID 'DEFINE LPAREN*
(tcp_rule | udp_rule | icmp_rule)
RPAREN SEMI
tcp_rule : "TCP" *direction src_addr dest_addr src_port dest_port*
connect action
udp_rule : "UDP" *direction src_addr dest_addr src_port dest_port action*
icmp_rule : "ICMP" *direction src_addr dest_addr type action*
direction : ("in" | "out")
src_addr : "src=" (*ipaddr* | *STAR*)
dest_addr : "dest=" (*ipaddr* | *STAR*)
src_port : "src_port=" ((*CMPR*)? *INT* | *STAR*)
dest_port : "src_port=" ((*CMPR*)? *INT* | *STAR*)
connect : *INT*
type : *INT*
action : ("allow" | "block")

loop : (*net_loop* | *host_loop* | *do_loop*)
net_loop : "for" "network" *LPAREN ID* ("to" *ID*)? *RPAREN*
LBRKT (*stmt*)* *RBRKT*
host_loop : "for" "host" *LPAREN ID* ("to" *ID*)? *RPAREN*
LBRKT (*stmt*)* *RBRKT*
do_loop : "do" *LPAREN ID EQ INT* ("to" *INT* ("by" *INT*)?)? *RPAREN*
LBRKT (*stmt*)* *RBRKT*

decision : (*net_if* | *host_if* | *var_if*)
net_if : "if" *LPAREN* "network" *CMPR ID RPAREN*
LBRKT (*stmt*)* *RBRKT* ("else" *LBRKT* (stmt)* *RBRKT*)?
host_if : "if" *LPAREN* "host" *CMPR ID RPAREN*
LBRKT (*stmt*)* *RBRKT* ("else" *LBRKT* (stmt)* *RBRKT*)?

var_if : "if" *LPAREN* ID *CMPR INT RPAREN*
LBRKT (*stmt*)* *RBRKT* ("else" *LBRKT* (stmt)* *RBRKT*)?

assignment : (*net_assign* | *host_assign* | *incr_add*)
net_assign : *ID H_ASSIGN ID (PLUS ID)* SEMI*
host_assign : *ID R_ASSIGN ID (PLUS ID)* SEMI*
incr_add : *ID INCR_ASSIGN ID SEMI*

probe : (*tcp_pkt* | *udp_pkt* | *icmp_pkt*)
tcp_pkt : "snd_tcp" *LPAREN* "src=" *v_ipaddr POUND v_int*
"dest=" *v_ipaddr POUND v_int*
"cntl=" *v_int*
RPAREN SEMI
udp_pkt : "snd_udp" *LPAREN* "src=" *v_ipaddr POUND v_int*
"dest=" *v_ipaddr POUND v_int*
RPAREN SEMI
icmp_pkt : "snd_icmp" *LPAREN* "src=" *v_ipaddr*
"dest=" *v_ipaddr*
"type=" *v_int*
RPAREN SEMI
v_ipaddr : *v_int DOT v_int DOT v_int DOT v_int*
v_int : (*INT* | *ID*)

report : "report" "target=" (*STAR* | "attack" | *ID*)
"protocol=" (*STAR* | "TCP" | "UDP" | "ICMP")
"direction=" (*STAR* | "snd" | "rcv")
SEMI

*ID: ALPHA (ALPHA | DIGIT | '_')**
ALPHA : ('a' .. 'z' | 'A'..'Z')

INT : (DIGIT)+
DIGIT : '0' ... '9'

DEFINE : ':='
EQ : '='
H_ASSIGN : '@='
R_ASSIGN : '<-'
VAR_ASSIGN : '='
CMPR : ('<' | '>' | '==' | '<=' | '>=')
INCR_ASSIGN : '+='

SEMI : ' '
COMMA : '\t'
COLON : '\n'
LPAREN : '('
RPAREN : ')'
LBRKT : '{'
RBRKT : '}'
SLASH : '/'
POUND : '#'
STAR : '*'
DOT : '.'

4.0 Project Plan

4.1 Planning Process

The planning process consisted of coming up with the phases required to realize the project. These fundamental steps were identified for progression:

- Language Design
- Data Structure Design
- Implementation Design
- Implementation
- Testing
- Documentation

4.2 Programming Style Guide

Programming style consisted of the following rules:

- Comments required for every class
- Short comment for every method in class
- For lengthy methods comment purpose of major blocks
- Indent java code segments in brackets and line up bracket start and end of code segment
- Utilize java.util data structures whenever possible

4.3 Software Development Environment

The development Java development environment under AIX was utilized.

4.4 Roles and Responsibilities

As sole developer, roles and responsibilities converged.

4.5 Project Time Line & Log

The following table encompasses my project time line and log.

Table 1: Project Time Line and Log

Project Function	Projected Start	Projected End	Actual Start	Actual End
Language Design	11/15/05	12/01/05	11/15/05	12/01/05
Data Structure Design	11/15/05	12/01/05	11/25/05	12/15/05
Implementation Design	12/01/05	12/15/05	12/01/05	12/15/05
Implementation	12/01/05	12/15/05	12/02/05	12/15/05
Testing	12/01/05	12/15/05	12/10/05	12/19/05
Documentation	12/01/20	12/19/05	12/10/05	12/20/05

5.0 Architectural Design

Figure 2 is a block diagram of the major software components and flow of the Firedrl language implementation. The Antlr tool is utilized to generate a Java version of a Lexer, Parser and AST Walker shown in the figure. The Firedrl input source code is fed into a compile version of these components. The major functionality of the Firedrl language resides in the TestNetWork object instantiated by Java code annotated to the AST walker. The TestNetWork methods provide the functionality to build (left blocks) and test (right blocks) the network image abstracted within the data objects instantiated in the TestNetWork class. The build methods consist of a group of network definition methods and network connection methods that configure the TestNetWork object with LANs, hosts, and rules with the proper connections and associations. The test methods consists of methods to drive test data and report information. The control methods influence the test sequence. The following are the major communication interfaces:

Build Methods

Definition Type

addRule -- Add rule definition
addLan -- Add LAN definition
addHost -- Add Host definition

Connection Type

addHostToLan -- Connect host to a LAN
addRuleToHost -- Connect rule to a host

Test Methods

Driver Type

Probe -- Sends pkt to all hosts which may be filtered by firewall rules
Report -- Enables hosts for reporting access events

Control Type

The control methods are directly implemented in annotated AST walker code.

v

5.1 Design Considerations

Most of the network components are implemented as Java HashTable constructs. Using the key/value paradigm, associations can be easily modeled in order to determine the characteristics of a network, LAN or host. Keys are used to reference related information in other objects in a similar manner as databases. For example, to probe all hosts on a Lan, the Lan object will contain just the host names (keys) which are utilized to access the particular hosts to gain further information such as network ID.

Type checking was dictated by the data organization of identifiers. This was done to facilitate type checking of assignments where the right side is of different type than the left side, while being precise in the context. Each type of identifier (e.g. LAN, host, rule, variable) were stored in separate data structures. Names are globally unique, such that all structures are checked to see if a newly defined identifier exits. An assignment statement type check only checks the data structures associated with the context. For example, hosts are assigned to LANs, such that Host types are assigned to a Lan type. This contextual requirement can be managed by checking the right side name for existence in the Lan names data structure, and the left side host names for existence in the Host names data structure.

The actual runtime processing was folded into the AST walker as additional methods. All host are interrogated to see if the network address is compatible. If that test is o.k., then the firewall rules are interrogated to determine if the packet is allowed through. The passage of a packet into a host is reported as an event if the report filter enables the host.

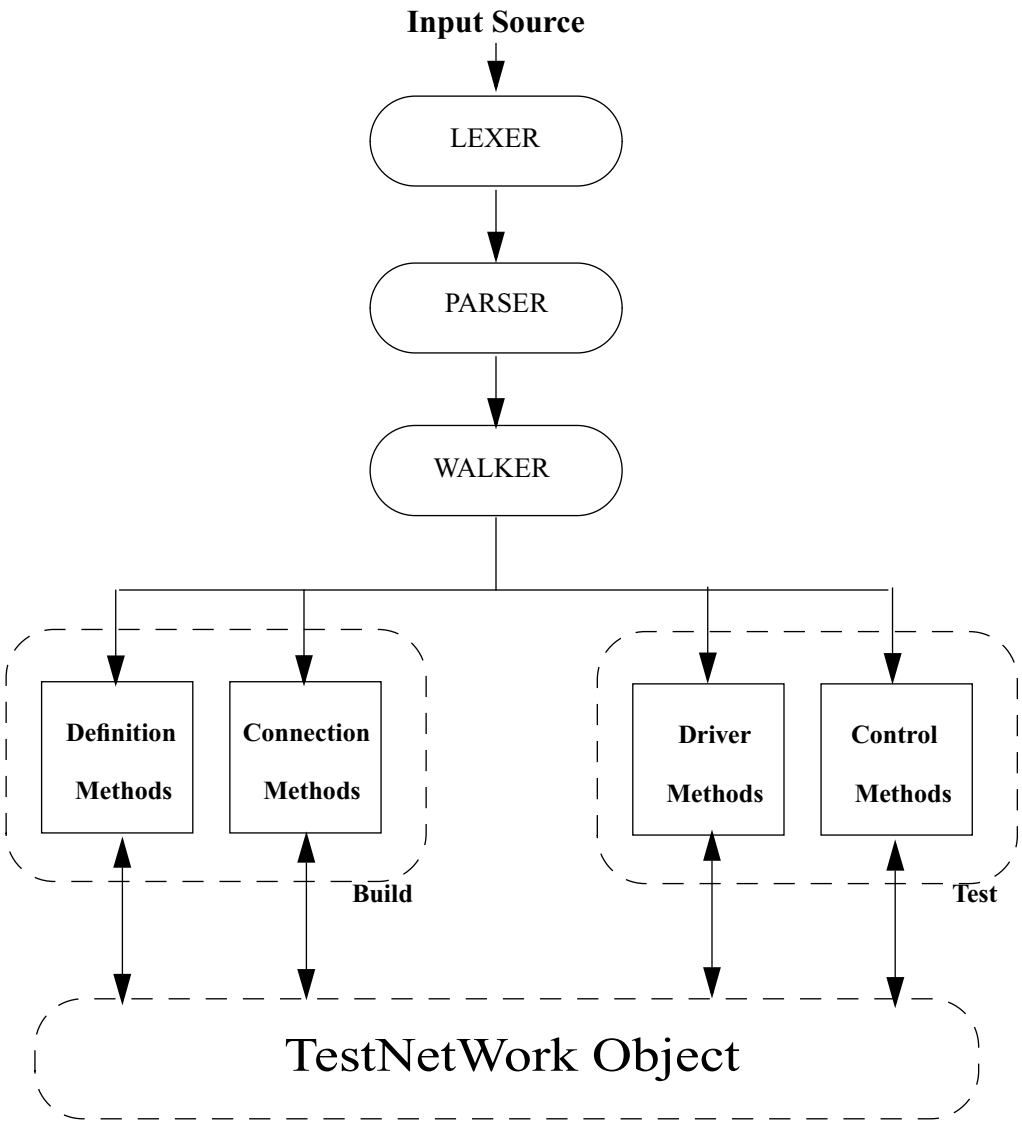

Figure 2: Firedrl Block Diagram

6.0 Test Plan

The test plan consisted of three phases. The object of phase 1 is to perform semantic checks to assure that Firedrl input source will reject input that is inconsistent. The object of phase 2 is to assure the network can be rendered accurately. The object of the third phase is to drive the network and generate output reports.

6.1 Phase 1: Semantic Checks

The following tests are typical of the set of semantic checks tested:

- Name doesn't exist on assignment
- Name exists already on definition
- Host assignment to Lan heterogeneous type check
- Rule assignment to Host heterogeneous type check

Here is an example of source with a semantic error in Host to Lan assignment, where a host name is erroneously used as a Lan name:

```
//-------------------------------------------
// Test Invalid Host Assignment
//-------------------------------------------
host_def A,B,C,INVAL;   // Define Host Name INVAL
lan_def Net1:10.2.4.3/16, Net2:11.2.4.2/24;
Net1 @= A:10, B:20;
INVAL @= C:30, D:5;     // Try to assign hosts to it
var a,b,c,d,e;
a = 5;
c = 3;
b = a;
rule Rule1 = block in  10.4.32.1 * TCP * 24 5;
rule Rule2 = allow out 10.3.4.5 * TCP * 24 5;
rule Rule3 = allow out 10.3.4.5 * UDP * 77;
A %= Rule1, Rule2;
B %= Rule2;
```

Here is the output from Firedrl:

```
Error: Name: INVAL not lan name type!
```

6.2 Phase 2: Network Checks

The following is the test plan to verify network object validity:

- Lan names recorded correctly
- Host names associated with LANs recorded correctly
- Rule definitions recorded correctly
- Host definitions recorded correctly

- Host interfaces on LANs defined
- Rules are associated with correct host

Here is an example of source which checks out a network definition:

```
//-------------------------------------------------
// Network Topology Validity Test
//-------------------------------------------------
//-- Define hosts and LANs
host_def A,B,C,D;
lan_def Net1:10.2.4.3/16, Net2:11.2.4.2/24;

//-- Set interface connectons
Net1 @= A:10, B:20;
Net2 @= C:30, D:5;

//-- Define rules
rule Rule1 = block in  10.4.32.1 * TCP * 24 5;
rule Rule2 = allow out 10.3.4.5  * TCP * 24 5;
rule Rule3 = allow out 10.3.4.5  * UDP * 77;

//-- Assign rules
A %= Rule1, Rule2;
B %= Rule2;
C %= Rule1,Rule3;

//-- Report Statement
report topology;
```

Here is output from Firedrl:

```
+-------------------------------
+ Lan Definitions
+-------------------------------
 Lan Name = Net2 IP Addr =  11.2.4.2 Net Msk =  24
  -- Host= D
  -- Host= C
 Lan Name = Net1 IP Addr =  10.2.4.3 Net Msk =  16
  -- Host= A
  -- Host= B
+-------------------------------
+ Host Definitions
+-------------------------------
 Host Name = A
 --- Lan Name = Net1
     Lan ID = 10
 --- Rule Name = Rule2
 --- Rule Name = Rule1
```

```
Host Name = D
--- Lan Name = Net2
    Lan ID = 5
Host Name = C
--- Lan Name = Net2
    Lan ID = 30
--- Rule Name = Rule3
--- Rule Name = Rule1
Host Name = B
--- Lan Name = Net1
    Lan ID = 20
--- Rule Name = Rule2
+------------------------------
+ Rule Definitions
+------------------------------
Rule name = Rule3
---- type = UDP act =  allow dir = out
     src =  10.3.4.5 s_port = * dest = * d_port = 77 i_type = 0 con =0
Rule name = Rule2
---- type = TCP act =  allow dir = out
     src =  10.3.4.5 s_port = * dest = * d_port = 24 i_type = 0 con =5
Rule name = Rule1
---- type = TCP act =  block dir = in
     src =  10.4.32.1 s_port = * dest = * d_port = 24 i_type = 0 con =5
```

6.3 Phase 3: Firewall Checks

The following is the test plan to verify network drive and firewall filtering:

- Lan specific firewall report
- Host specific firewall report
- TCP specific firewall report
- UDP specific firewall report
- ICMP specific firewall report
- Input Only firewall report

7.0 Lessons Learned

Allow plenty of test time.

Teammates are important for feedback and load sharing.

Finish the easy stuff first (e.g. don't consume time and testing of most difficult items at the expense of finishing well defined items.)

Design in a debug framework prior to testing, such that pertinent information needed for debug is factored into the schedule.

8.0 Appendix

```
//Marvin Rich -- Lexer
// $ANTLR 2.7.5 (20050128): “Firedrl.g” -> “FiredrlLexer.java”$

import java.io.InputStream;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;
import antlr.TokenStreamRecognitionException;
import antlr.CharStreamException;
import antlr.CharStreamIOException;
import antlr.ANTLRException;
import java.io.Reader;
import java.util.Hashtable;
import antlr.CharScanner;
import antlr.InputBuffer;
import antlr.ByteBuffer;
import antlr.CharBuffer;
import antlr.Token;
import antlr.CommonToken;
import antlr.RecognitionException;
import antlr.NoViableAltForCharException;
import antlr.MismatchedCharException;
import antlr.TokenStream;
import antlr.ANTLRHashString;
import antlr.LexerSharedInputState;
import antlr.collections.impl.BitSet;
import antlr.SemanticException;

public class FiredrlLexer extends antlr.CharScanner implements FiredrlToken-
Types, TokenStream
 {
public FiredrlLexer(InputStream in) {
this(new ByteBuffer(in));
}
public FiredrlLexer(Reader in) {
this(new CharBuffer(in));
}
public FiredrlLexer(InputBuffer ib) {
this(new LexerSharedInputState(ib));
}
public FiredrlLexer(LexerSharedInputState state) {
super(state);
caseSensitiveLiterals = true;
setCaseSensitive(true);
literals = new Hashtable();
```

```
literals.put(new ANTLRHashString("for", this), new Integer(34));
literals.put(new ANTLRHashString("if", this), new Integer(32));
literals.put(new ANTLRHashString("rcv", this), new Integer(56));
literals.put(new ANTLRHashString("snd", this), new Integer(55));
literals.put(new ANTLRHashString("direction=", this), new Integer(50));
literals.put(new ANTLRHashString("target=", this), new Integer(48));
literals.put(new ANTLRHashString("UDP", this), new Integer(45));
literals.put(new ANTLRHashString("@=", this), new Integer(40));
literals.put(new ANTLRHashString("var", this), new Integer(39));
literals.put(new ANTLRHashString("endif", this), new Integer(31));
literals.put(new ANTLRHashString("%=", this), new Integer(41));
literals.put(new ANTLRHashString("<>", this), new Integer(42));
literals.put(new ANTLRHashString(">", this), new Integer(57));
literals.put(new ANTLRHashString("out", this), new Integer(54));
literals.put(new ANTLRHashString("then", this), new Integer(33));
literals.put(new ANTLRHashString("HOSTS", this), new Integer(35));
literals.put(new ANTLRHashString("rule", this), new Integer(43));
literals.put(new ANTLRHashString("in", this), new Integer(36));
literals.put(new ANTLRHashString("report", this), new Integer(47));
literals.put(new ANTLRHashString("protocol=", this), new Integer(49));
literals.put(new ANTLRHashString("lan_def", this), new Integer(37));
literals.put(new ANTLRHashString("ICMP", this), new Integer(46));
literals.put(new ANTLRHashString("allow", this), new Integer(52));
literals.put(new ANTLRHashString("host_def", this), new Integer(38));
literals.put(new ANTLRHashString("block", this), new Integer(53));
literals.put(new ANTLRHashString("TCP", this), new Integer(44));
literals.put(new ANTLRHashString("topology", this), new Integer(51));
}

public Token nextToken() throws TokenStreamException {
Token theRetToken=null;
tryAgain:
for (;;) {
Token _token = null;
int _ttype = Token.INVALID_TYPE;
resetText();
try {   // for char stream error handling
try {   // for lexical error handling
switch ( LA(1)) {
case ':':
{
mCOLON(true);
theRetToken=_returnToken;
break;
}
case '.':
```

```
{
mDOT(true);
theRetToken=_returnToken;
break;
}
case '+':
{
mPLUS(true);
theRetToken=_returnToken;
break;
}
case '*':
{
mSTAR(true);
theRetToken=_returnToken;
break;
}
case ',':
{
mCOMMA(true);
theRetToken=_returnToken;
break;
}
case '(':
{
mLPAREN(true);
theRetToken=_returnToken;
break;
}
case ')':
{
mRPAREN(true);
theRetToken=_returnToken;
break;
}
case '{':
{
mLBRKT(true);
theRetToken=_returnToken;
break;
}
case '}':
{
mRBRKT(true);
theRetToken=_returnToken;
break;
```

```
}
case '@':
{
mNEQ(true);
theRetToken=_returnToken;
break;
}
case '%':
{
mREQ(true);
theRetToken=_returnToken;
break;
}
case ';':
{
mSEMI(true);
theRetToken=_returnToken;
break;
}
case 'A': case 'B': case 'C': case 'D':
case 'E': case 'F': case 'G': case 'H':
case 'I': case 'J': case 'K': case 'L':
case 'M': case 'N': case 'O': case 'P':
case 'Q': case 'R': case 'S': case 'T':
case 'U': case 'V': case 'W': case 'X':
case 'Y': case 'Z': case 'a': case 'b':
case 'c': case 'd': case 'e': case 'f':
case 'g': case 'h': case 'i': case 'j':
case 'k': case 'l': case 'm': case 'n':
case 'o': case 'p': case 'q': case 'r':
case 's': case 't': case 'u': case 'v':
case 'w': case 'x': case 'y': case 'z':
{
mID(true);
theRetToken=_returnToken;
break;
}
case '0': case '1': case '2': case '3':
case '4': case '5': case '6': case '7':
case '8': case '9':
{
mINT(true);
theRetToken=_returnToken;
break;
}
case '':
```

```
{
mSTRING(true);
theRetToken=_returnToken;
break;
}
case '\t':  case '\n':  case '\r':  case ' ':
{
mWS(true);
theRetToken=_returnToken;
break;
}
default:
if ((LA(1)=='=') && (LA(2)=='=')) {
mEQ(true);
theRetToken=_returnToken;
}
else if ((LA(1)=='<') && (LA(2)=='=')) {
mLE(true);
theRetToken=_returnToken;
}
else if ((LA(1)=='>') && (LA(2)=='=')) {
mGE(true);
theRetToken=_returnToken;
}
else if ((LA(1)=='<') && (LA(2)=='>')) {
mHASSIGN(true);
theRetToken=_returnToken;
}
else if ((LA(1)=='/') && (LA(2)=='/')) {
mCMNT(true);
theRetToken=_returnToken;
}
else if ((LA(1)=='/') && (true)) {
mSLASH(true);
theRetToken=_returnToken;
}
else if ((LA(1)=='=') && (true)) {
mASSIGN(true);
theRetToken=_returnToken;
}
else if ((LA(1)=='>') && (true)) {
mGT(true);
theRetToken=_returnToken;
}
else if ((LA(1)=='<') && (true)) {
mLT(true);
```

```
theRetToken=_returnToken;
}
else {
if (LA(1)==EOF_CHAR) {uponEOF(); _returnToken = makeTo-
ken(Token.EOF_TYPE);}
else {throw new NoViableAltForCharException((char)LA(1), getFilename(), get-
Line(), getColumn());}
}
}
if ( _returnToken==null ) continue tryAgain; // found SKIP token
_ttype = _returnToken.getType();
_ttype = testLiteralsTable(_ttype);
_returnToken.setType(_ttype);
return _returnToken;
}
catch (RecognitionException e) {
throw new TokenStreamRecognitionException(e);
}
}
catch (CharStreamException cse) {
if ( cse instanceof CharStreamIOException ) {
throw new TokenStreamIOException(((CharStreamIOException)cse).io);
}
else {
throw new TokenStreamException(cse.getMessage());
}
}
}
}

public final void mCOLON(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = COLON;
int _saveIndex;

match(':');
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mSLASH(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
```

```
int _ttype; Token _token=null; int _begin=text.length();
_ttype = SLASH;
int _saveIndex;

match('/');
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mDOT(boolean _createToken) throws RecognitionException, Char-
StreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = DOT;
int _saveIndex;

match('.');
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mPLUS(boolean _createToken) throws RecognitionException, Char-
StreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = PLUS;
int _saveIndex;

match('+');
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mSTAR(boolean _createToken) throws RecognitionException, Char-
StreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = STAR;
int _saveIndex;
```

```
match('*');
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mASSIGN(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = ASSIGN;
int _saveIndex;

match('=');
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mCOMMA(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = COMMA;
int _saveIndex;

match(',');
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mLPAREN(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = LPAREN;
int _saveIndex;

match('(');
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
```

```
}
_returnToken = _token;
}

public final void mRPAREN(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = RPAREN;
int _saveIndex;

match(')');
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mLBRKT(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = LBRKT;
int _saveIndex;

match('{');
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mRBRKT(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = RBRKT;
int _saveIndex;

match('}');
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}
```

```
public final void mEQ(boolean _createToken) throws RecognitionException, Char-
StreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = EQ;
int _saveIndex;

match("==");
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mLE(boolean _createToken) throws RecognitionException, Char-
StreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = LE;
int _saveIndex;

match("<=");
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mGE(boolean _createToken) throws RecognitionException, Char-
StreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = GE;
int _saveIndex;

match(">=");
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mGT(boolean _createToken) throws RecognitionException, Char-
StreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = GT;
```

```
int _saveIndex;

match('>');
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mLT(boolean _createToken) throws RecognitionException, Char-
StreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = LT;
int _saveIndex;

match('<');
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mNEQ(boolean _createToken) throws RecognitionException, Char-
StreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = NEQ;
int _saveIndex;

match("@=");
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mREQ(boolean _createToken) throws RecognitionException, Char-
StreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = REQ;
int _saveIndex;

match("%=");
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
```

```
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mHASSIGN(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = HASSIGN;
int _saveIndex;

match("<>");
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mSEMI(boolean _createToken) throws RecognitionException, Char-
StreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = SEMI;
int _saveIndex;

match(';');
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

protected final void mLETTER(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = LETTER;
int _saveIndex;

switch ( LA(1)) {
case 'a':  case 'b':  case 'c':  case 'd':
case 'e':  case 'f':  case 'g':  case 'h':
case 'i':  case 'j':  case 'k':  case 'l':
case 'm':  case 'n':  case 'o':  case 'p':
case 'q':  case 'r':  case 's':  case 't':
```

```
case ‘u’:  case ‘v’:  case ‘w’:  case ‘x’:
case ‘y’:  case ‘z’:
{
{
matchRange(‘a’,’z’);
}
break;
}
case ‘A’:  case ‘B’:  case ‘C’:  case ‘D’:
case ‘E’:  case ‘F’:  case ‘G’:  case ‘H’:
case ‘I’:  case ‘J’:  case ‘K’:  case ‘L’:
case ‘M’:  case ‘N’:  case ‘O’:  case ‘P’:
case ‘Q’:  case ‘R’:  case ‘S’:  case ‘T’:
case ‘U’:  case ‘V’:  case ‘W’:  case ‘X’:
case ‘Y’:  case ‘Z’:
{
{
matchRange(‘A’,’Z’);
}
break;
}
default:
{
throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine(), get-
Column());
}
}
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

protected final void mDIGIT(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = DIGIT;
int _saveIndex;

matchRange(‘0’,’9’);
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
```

```
}

public final void mID(boolean _createToken) throws RecognitionException, Char-
StreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = ID;
int _saveIndex;

mLETTER(false);
{
_loop27:
do {
switch ( LA(1)) {
case 'A': case 'B': case 'C': case 'D':
case 'E': case 'F': case 'G': case 'H':
case 'I': case 'J': case 'K': case 'L':
case 'M': case 'N': case 'O': case 'P':
case 'Q': case 'R': case 'S': case 'T':
case 'U': case 'V': case 'W': case 'X':
case 'Y': case 'Z': case 'a': case 'b':
case 'c': case 'd': case 'e': case 'f':
case 'g': case 'h': case 'i': case 'j':
case 'k': case 'l': case 'm': case 'n':
case 'o': case 'p': case 'q': case 'r':
case 's': case 't': case 'u': case 'v':
case 'w': case 'x': case 'y': case 'z':
{
mLETTER(false);
break;
}
case '0': case '1': case '2': case '3':
case '4': case '5': case '6': case '7':
case '8': case '9':
{
mDIGIT(false);
break;
}
case '_':
{
match('_');
break;
}
default:
{
break _loop27;
}
```

```
}
} while (true);
}
_ttype = testLiteralsTable(_ttype);
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mINT(boolean _createToken) throws RecognitionException, Char-
StreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = INT;
int _saveIndex;

{
int _cnt30=0;
_loop30:
do {
if (((LA(1) >= '0' && LA(1) <= '9'))) {
mDIGIT(false);
}
else {
if ( _cnt30>=1 ) { break _loop30; } else {throw new NoViableAltForCharExcep-
tion((char)LA(1), getFilename(), getLine(), getColumn());}
}

_cnt30++;
} while (true);
}
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mSTRING(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = STRING;
int _saveIndex;

_saveIndex=text.length();
```

```
match("");
text.setLength(_saveIndex);
{
_loop34:
do {
if ((LA(1)=="") && (LA(2)="")) {
match("");
_saveIndex=text.length();
match("");
text.setLength(_saveIndex);
}
else if ((_tokenSet_0.member(LA(1)))) {
{
match(_tokenSet_0);
}
}
else {
break _loop34;
}

} while (true);
}
_saveIndex=text.length();
match("");
text.setLength(_saveIndex);
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mCMNT(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = CMNT;
int _saveIndex;

match("//");
{
_loop37:
do {
if ((_tokenSet_1.member(LA(1)))) {
matchNot('\n');
}
else {
```

```
break _loop37;
}

} while (true);
}
match('\n');
newline(); _ttype = Token.SKIP;
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mWS(boolean _createToken) throws RecognitionException, Char-
StreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = WS;
int _saveIndex;

{
switch ( LA(1)) {
case ' ':
{
match(' ');
break;
}
case '\t':
{
match('\t');
break;
}
case '\n':
{
match('\n');
newline();
break;
}
case '\r':
{
match('\r');
break;
}
default:
{
```

```
throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine(), get-
Column());
}
}
}
_ttype = Token.SKIP;
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}


private static final long[] mk_tokenSet_0() {
long[] data = { -17179869185L, -1L, 0L, 0L};
return data;
}
public static final BitSet _tokenSet_0 = new BitSet(mk_tokenSet_0());
private static final long[] mk_tokenSet_1() {
long[] data = { -1025L, -1L, 0L, 0L};
return data;
}
public static final BitSet _tokenSet_1 = new BitSet(mk_tokenSet_1());

}

//--- Marvin Rich  Parser
// $ANTLR 2.7.5 (20050128): “Firedrl.g” -> “FiredrlParser.java”$

import antlr.TokenBuffer;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;
import antlr.ANTLRException;
import antlr.LLkParser;
import antlr.Token;
import antlr.TokenStream;
import antlr.RecognitionException;
import antlr.NoViableAltException;
import antlr.MismatchedTokenException;
import antlr.SemanticException;
import antlr.ParserSharedInputState;
import antlr.collections.impl.BitSet;
import antlr.collections.AST;
import java.util.Hashtable;
import antlr.ASTFactory;
```

```
import antlr.ASTPair;
import antlr.collections.impl.ASTArray;

public class FiredrlParser extends antlr.LLkParser       implements FiredrlToken-
Types
 {

protected FiredrlParser(TokenBuffer tokenBuf, int k) {
  super(tokenBuf,k);
  tokenNames = _tokenNames;
  buildTokenTypeASTClassMap();
  astFactory = new ASTFactory(getTokenTypeToASTClassMap());
}

public FiredrlParser(TokenBuffer tokenBuf) {
  this(tokenBuf,2);
}

protected FiredrlParser(TokenStream lexer, int k) {
  super(lexer,k);
  tokenNames = _tokenNames;
  buildTokenTypeASTClassMap();
  astFactory = new ASTFactory(getTokenTypeToASTClassMap());
}

public FiredrlParser(TokenStream lexer) {
  this(lexer,2);
}

public FiredrlParser(ParserSharedInputState state) {
  super(state,2);
  tokenNames = _tokenNames;
  buildTokenTypeASTClassMap();
  astFactory = new ASTFactory(getTokenTypeToASTClassMap());
}

public final void file() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST file_AST = null;

try {      // for error handling
src_code();
astFactory.addASTChild(currentAST, returnAST);
file_AST = (AST)currentAST.root;
```

```
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_0);
}
returnAST = file_AST;
}

public final void src_code() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST src_code_AST = null;

try {      // for error handling
{
switch ( LA(1)) {
case EOF:
{
match(Token.EOF_TYPE);
break;
}
case ID:
case LITERAL_lan_def:
case LITERAL_host_def:
case LITERAL_var:
case 42:
case LITERAL_rule:
case LITERAL_report:
{
stmt();
astFactory.addASTChild(currentAST, returnAST);
match(SEMI);
src_code();
astFactory.addASTChild(currentAST, returnAST);
break;
}
case LITERAL_if:
case LITERAL_for:
{
cntl_stmt();
astFactory.addASTChild(currentAST, returnAST);
break;
}
case RBRKT:
{
```

```
AST tmp30_AST = null;
tmp30_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp30_AST);
match(RBRKT);
src_code();
astFactory.addASTChild(currentAST, returnAST);
break;
}
case LITERAL_endif:
{
match(LITERAL_endif);
match(SEMI);
src_code();
astFactory.addASTChild(currentAST, returnAST);
break;
}
default:
{
throw new NoViableAltException(LT(1), getFilename());
}
}
}
src_code_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_0);
}
returnAST = src_code_AST;
}

public final void stmt() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST stmt_AST = null;

try {      // for error handling
switch ( LA(1)) {
case LITERAL_lan_def:
{
AST tmp33_AST = null;
tmp33_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp33_AST);
match(LITERAL_lan_def);
lan();
```

```
astFactory.addASTChild(currentAST, returnAST);
{
_loop46:
do {
if ((LA(1)==COMMA)) {
match(COMMA);
lan();
astFactory.addASTChild(currentAST, returnAST);
}
else {
break _loop46;
}

} while (true);
}
stmt_AST = (AST)currentAST.root;
break;
}
case LITERAL_host_def:
{
AST tmp35_AST = null;
tmp35_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp35_AST);
match(LITERAL_host_def);
AST tmp36_AST = null;
tmp36_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp36_AST);
match(ID);
{
_loop48:
do {
if ((LA(1)==COMMA)) {
match(COMMA);
AST tmp38_AST = null;
tmp38_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp38_AST);
match(ID);
}
else {
break _loop48;
}

} while (true);
}
stmt_AST = (AST)currentAST.root;
break;
```

```
}
case LITERAL_var:
{
AST tmp39_AST = null;
tmp39_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp39_AST);
match(LITERAL_var);
AST tmp40_AST = null;
tmp40_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp40_AST);
match(ID);
{
_loop50:
do {
if ((LA(1)==COMMA)) {
match(COMMA);
AST tmp42_AST = null;
tmp42_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp42_AST);
match(ID);
}
else {
break _loop50;
}

} while (true);
}
stmt_AST = (AST)currentAST.root;
break;
}
case 42:
{
AST tmp43_AST = null;
tmp43_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp43_AST);
match(42);
AST tmp44_AST = null;
tmp44_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp44_AST);
match(ID);
{
_loop56:
do {
if ((LA(1)==COMMA)) {
match(COMMA);
AST tmp46_AST = null;
```

```
tmp46_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp46_AST);
match(ID);
}
else {
break _loop56;
}

} while (true);
}
stmt_AST = (AST)currentAST.root;
break;
}
case LITERAL_rule:
{
match(LITERAL_rule);
AST tmp48_AST = null;
tmp48_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp48_AST);
match(ID);
match(ASSIGN);
action();
astFactory.addASTChild(currentAST, returnAST);
direction();
astFactory.addASTChild(currentAST, returnAST);
ip_sel();
astFactory.addASTChild(currentAST, returnAST);
ip_sel();
astFactory.addASTChild(currentAST, returnAST);
{
switch ( LA(1)) {
case LITERAL_TCP:
{
{
AST tmp50_AST = null;
tmp50_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp50_AST);
match(LITERAL_TCP);
{
port();
astFactory.addASTChild(currentAST, returnAST);
}
{
port();
astFactory.addASTChild(currentAST, returnAST);
}
```

```
AST tmp51_AST = null;
tmp51_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp51_AST);
match(INT);
}
break;
}
case LITERAL_UDP:
{
{
AST tmp52_AST = null;
tmp52_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp52_AST);
match(LITERAL_UDP);
{
port();
astFactory.addASTChild(currentAST, returnAST);
}
{
port();
astFactory.addASTChild(currentAST, returnAST);
}
}
break;
}
case LITERAL_ICMP:
{
{
AST tmp53_AST = null;
tmp53_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp53_AST);
match(LITERAL_ICMP);
AST tmp54_AST = null;
tmp54_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp54_AST);
match(INT);
}
break;
}
default:
{
throw new NoViableAltException(LT(1), getFilename());
}
}
}
stmt_AST = (AST)currentAST.root;
```

```
break;
}
default:
if ((LA(1)==ID) && (LA(2)==40)) {
AST tmp55_AST = null;
tmp55_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp55_AST);
match(ID);
AST tmp56_AST = null;
tmp56_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp56_AST);
match(40);
host_intfc();
astFactory.addASTChild(currentAST, returnAST);
{
_loop52:
do {
if ((LA(1)==COMMA)) {
match(COMMA);
host_intfc();
astFactory.addASTChild(currentAST, returnAST);
}
else {
break _loop52;
}

} while (true);
}
stmt_AST = (AST)currentAST.root;
}
else if ((LA(1)==ID) && (LA(2)==41)) {
AST tmp58_AST = null;
tmp58_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp58_AST);
match(ID);
AST tmp59_AST = null;
tmp59_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp59_AST);
match(41);
AST tmp60_AST = null;
tmp60_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp60_AST);
match(ID);
{
_loop54:
do {
```

```
if ((LA(1)==COMMA)) {
match(COMMA);
AST tmp62_AST = null;
tmp62_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp62_AST);
match(ID);
}
else {
break _loop54;
}

} while (true);
}
stmt_AST = (AST)currentAST.root;
}
else if ((LA(1)==ID) && (LA(2)==ASSIGN)) {
AST tmp63_AST = null;
tmp63_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp63_AST);
match(ID);
AST tmp64_AST = null;
tmp64_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp64_AST);
match(ASSIGN);
var_src();
astFactory.addASTChild(currentAST, returnAST);
stmt_AST = (AST)currentAST.root;
}
else if ((LA(1)==LITERAL_report) && (LA(2)==48)) {
AST tmp65_AST = null;
tmp65_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp65_AST);
match(LITERAL_report);
match(48);
rpt_targ();
astFactory.addASTChild(currentAST, returnAST);
match(49);
rpt_proto();
astFactory.addASTChild(currentAST, returnAST);
match(50);
rpt_dir();
astFactory.addASTChild(currentAST, returnAST);
stmt_AST = (AST)currentAST.root;
}
else if ((LA(1)==LITERAL_report) && (LA(2)==LITERAL_topology)) {
AST tmp69_AST = null;
```

```
tmp69_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp69_AST);
match(LITERAL_report);
AST tmp70_AST = null;
tmp70_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp70_AST);
match(LITERAL_topology);
stmt_AST = (AST)currentAST.root;
}
else {
throw new NoViableAltException(LT(1), getFilename());
}
}
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_1);
}
returnAST = stmt_AST;
}

public final void cntl_stmt() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST cntl_stmt_AST = null;

try {      // for error handling
switch ( LA(1)) {
case LITERAL_if:
{
AST tmp71_AST = null;
tmp71_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp71_AST);
match(LITERAL_if);
pred();
astFactory.addASTChild(currentAST, returnAST);
match(LITERAL_then);
src_code();
astFactory.addASTChild(currentAST, returnAST);
cntl_stmt_AST = (AST)currentAST.root;
break;
}
case LITERAL_for:
{
AST tmp73_AST = null;
```

```
tmp73_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp73_AST);
match(LITERAL_for);
match(LPAREN);
match(LITERAL_HOSTS);
match(LITERAL_in);
AST tmp77_AST = null;
tmp77_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp77_AST);
match(ID);
match(RPAREN);
match(LBRKT);
src_code();
astFactory.addASTChild(currentAST, returnAST);
cntl_stmt_AST = (AST)currentAST.root;
break;
}
default:
{
throw new NoViableAltException(LT(1), getFilename());
}
}
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_0);
}
returnAST = cntl_stmt_AST;
}

public final void pred() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST pred_AST = null;

try {      // for error handling
AST tmp80_AST = null;
tmp80_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp80_AST);
match(ID);
AST tmp81_AST = null;
tmp81_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp81_AST);
match(57);
AST tmp82_AST = null;
```

```
tmp82_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp82_AST);
match(INT);
pred_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_2);
}
returnAST = pred_AST;
}

public final void lan() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST lan_AST = null;

try {      // for error handling
AST tmp83_AST = null;
tmp83_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp83_AST);
match(ID);
AST tmp84_AST = null;
tmp84_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp84_AST);
match(COLON);
ipaddr();
astFactory.addASTChild(currentAST, returnAST);
match(SLASH);
AST tmp86_AST = null;
tmp86_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp86_AST);
match(INT);
lan_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_3);
}
returnAST = lan_AST;
}

public final void host_intfc() throws RecognitionException, TokenStreamException {

returnAST = null;
```

```
ASTPair currentAST = new ASTPair();
AST host_intfc_AST = null;

try {      // for error handling
AST tmp87_AST = null;
tmp87_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp87_AST);
match(ID);
match(COLON);
AST tmp89_AST = null;
tmp89_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp89_AST);
match(INT);
host_intfc_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_3);
}
returnAST = host_intfc_AST;
}

public final void var_src() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST var_src_AST = null;

try {      // for error handling
{
switch ( LA(1)) {
case ID:
{
AST tmp90_AST = null;
tmp90_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp90_AST);
match(ID);
break;
}
case INT:
{
AST tmp91_AST = null;
tmp91_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp91_AST);
match(INT);
break;
```

```
}
default:
{
throw new NoViableAltException(LT(1), getFilename());
}
}
}
var_src_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_1);
}
returnAST = var_src_AST;
}

public final void action() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST action_AST = null;

try {      // for error handling
{
switch ( LA(1)) {
case LITERAL_allow:
{
AST tmp92_AST = null;
tmp92_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp92_AST);
match(LITERAL_allow);
break;
}
case LITERAL_block:
{
AST tmp93_AST = null;
tmp93_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp93_AST);
match(LITERAL_block);
break;
}
default:
{
throw new NoViableAltException(LT(1), getFilename());
}
}
```

```
}
action_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_4);
}
returnAST = action_AST;
}

public final void direction() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST direction_AST = null;

try {      // for error handling
{
switch ( LA(1)) {
case LITERAL_in:
{
AST tmp94_AST = null;
tmp94_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp94_AST);
match(LITERAL_in);
break;
}
case LITERAL_out:
{
AST tmp95_AST = null;
tmp95_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp95_AST);
match(LITERAL_out);
break;
}
default:
{
throw new NoViableAltException(LT(1), getFilename());
}
}
}
direction_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_5);
```

```
}
returnAST = direction_AST;
}

public final void ip_sel() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST ip_sel_AST = null;

try {      // for error handling
{
switch ( LA(1)) {
case INT:
{
ipaddr();
astFactory.addASTChild(currentAST, returnAST);
break;
}
case STAR:
{
AST tmp96_AST = null;
tmp96_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp96_AST);
match(STAR);
break;
}
default:
{
throw new NoViableAltException(LT(1), getFilename());
}
}
}
ip_sel_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_6);
}
returnAST = ip_sel_AST;
}

public final void port() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
```

```
AST port_AST = null;

try {      // for error handling
{
switch ( LA(1)) {
case EQ:
case LE:
case GE:
case GT:
case LT:
case INT:
{
{
switch ( LA(1)) {
case EQ:
case LE:
case GE:
case GT:
case LT:
{
cmpr();
astFactory.addASTChild(currentAST, returnAST);
break;
}
case INT:
{
break;
}
default:
{
throw new NoViableAltException(LT(1), getFilename());
}
}
}
AST tmp97_AST = null;
tmp97_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp97_AST);
match(INT);
break;
}
case STAR:
{
AST tmp98_AST = null;
tmp98_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp98_AST);
match(STAR);
```

```
break;
}
default:
{
throw new NoViableAltException(LT(1), getFilename());
}
}
}
port_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_7);
}
returnAST = port_AST;
}

public final void rpt_targ() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST rpt_targ_AST = null;

try {      // for error handling
{
switch ( LA(1)) {
case STAR:
{
AST tmp99_AST = null;
tmp99_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp99_AST);
match(STAR);
break;
}
case ID:
{
AST tmp100_AST = null;
tmp100_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp100_AST);
match(ID);
break;
}
default:
{
throw new NoViableAltException(LT(1), getFilename());
}
```

```
}
}
rpt_targ_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_8);
}
returnAST = rpt_targ_AST;
}

public final void rpt_proto() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST rpt_proto_AST = null;

try {      // for error handling
{
switch ( LA(1)) {
case STAR:
{
AST tmp101_AST = null;
tmp101_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp101_AST);
match(STAR);
break;
}
case LITERAL_TCP:
{
AST tmp102_AST = null;
tmp102_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp102_AST);
match(LITERAL_TCP);
break;
}
case LITERAL_UDP:
{
AST tmp103_AST = null;
tmp103_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp103_AST);
match(LITERAL_UDP);
break;
}
case LITERAL_ICMP:
{
```

```
AST tmp104_AST = null;
tmp104_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp104_AST);
match(LITERAL_ICMP);
break;
}
default:
{
throw new NoViableAltException(LT(1), getFilename());
}
}
}
rpt_proto_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_9);
}
returnAST = rpt_proto_AST;
}

public final void rpt_dir() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST rpt_dir_AST = null;

try {      // for error handling
{
switch ( LA(1)) {
case STAR:
{
AST tmp105_AST = null;
tmp105_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp105_AST);
match(STAR);
break;
}
case LITERAL_snd:
{
AST tmp106_AST = null;
tmp106_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp106_AST);
match(LITERAL_snd);
break;
}
```

```
case LITERAL_rcv:
{
AST tmp107_AST = null;
tmp107_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp107_AST);
match(LITERAL_rcv);
break;
}
default:
{
throw new NoViableAltException(LT(1), getFilename());
}
}
}
rpt_dir_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_1);
}
returnAST = rpt_dir_AST;
}

public final void ipaddr() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST ipaddr_AST = null;

try {      // for error handling
AST tmp108_AST = null;
tmp108_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp108_AST);
match(INT);
AST tmp109_AST = null;
tmp109_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp109_AST);
match(DOT);
AST tmp110_AST = null;
tmp110_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp110_AST);
match(INT);
match(DOT);
AST tmp112_AST = null;
tmp112_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp112_AST);
```

```
match(INT);
match(DOT);
AST tmp114_AST = null;
tmp114_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp114_AST);
match(INT);
ipaddr_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_10);
}
returnAST = ipaddr_AST;
}

public final void cmpr() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST cmpr_AST = null;

try {      // for error handling
switch ( LA(1)) {
case LT:
{
AST tmp115_AST = null;
tmp115_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp115_AST);
match(LT);
cmpr_AST = (AST)currentAST.root;
break;
}
case GT:
{
AST tmp116_AST = null;
tmp116_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp116_AST);
match(GT);
cmpr_AST = (AST)currentAST.root;
break;
}
case EQ:
{
AST tmp117_AST = null;
tmp117_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp117_AST);
```

```
match(EQ);
cmpr_AST = (AST)currentAST.root;
break;
}
case LE:
{
AST tmp118_AST = null;
tmp118_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp118_AST);
match(LE);
cmpr_AST = (AST)currentAST.root;
break;
}
case GE:
{
AST tmp119_AST = null;
tmp119_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp119_AST);
match(GE);
cmpr_AST = (AST)currentAST.root;
break;
}
default:
{
throw new NoViableAltException(LT(1), getFilename());
}
}
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_11);
}
returnAST = cmpr_AST;
}


public static final String[] _tokenNames = {
“<0>”,
“EOF”,
“<2>”,
“NULL_TREE_LOOKAHEAD”,
“COLON”,
“SLASH”,
“DOT”,
“PLUS”,
“STAR”,
```

“ASSIGN”,
“COMMA”,
“LPAREN”,
“RPAREN”,
“LBRKT”,
“RBRKT”,
“EQ”,
“LE”,
“GE”,
“GT”,
“LT”,
“NEQ”,
“REQ”,
“HASSIGN”,
“SEMI”,
“LETTER”,
“DIGIT”,
“ID”,
“INT”,
“STRING”,
“CMNT”,
“WS”,
“\”endif\””,
“\”if\””,
“\”then\””,
“\”for\””,
“\”HOSTS\””,
“\”in\””,
“\”lan_def\””,
“\”host_def\””,
“\”var\””,
“\”@=\””,
“\”%=\””,
“\”<>\””,
“\”rule\””,
“\”TCP\””,
“\”UDP\””,
“\”ICMP\””,
“\”report\””,
“\”target=\””,
“\”protocol=\””,
“\”direction=\””,
“\”topology\””,
“\”allow\””,
“\”block\””,
“\”out\””,

```
"\"snd\"",
"\"rcv\"",
"\">\""
};

protected void buildTokenTypeASTClassMap() {
tokenTypeToASTClassMap=null;
};

private static final long[] mk_tokenSet_0() {
long[] data = { 2L, 0L};
return data;
}
public static final BitSet _tokenSet_0 = new BitSet(mk_tokenSet_0());
private static final long[] mk_tokenSet_1() {
long[] data = { 8388608L, 0L};
return data;
}
public static final BitSet _tokenSet_1 = new BitSet(mk_tokenSet_1());
private static final long[] mk_tokenSet_2() {
long[] data = { 8589934592L, 0L};
return data;
}
public static final BitSet _tokenSet_2 = new BitSet(mk_tokenSet_2());
private static final long[] mk_tokenSet_3() {
long[] data = { 8389632L, 0L};
return data;
}
public static final BitSet _tokenSet_3 = new BitSet(mk_tokenSet_3());
private static final long[] mk_tokenSet_4() {
long[] data = { 18014467228958720L, 0L};
return data;
}
public static final BitSet _tokenSet_4 = new BitSet(mk_tokenSet_4());
private static final long[] mk_tokenSet_5() {
long[] data = { 134217984L, 0L};
return data;
}
public static final BitSet _tokenSet_5 = new BitSet(mk_tokenSet_5());
private static final long[] mk_tokenSet_6() {
long[] data = { 123145436528896L, 0L};
return data;
}
public static final BitSet _tokenSet_6 = new BitSet(mk_tokenSet_6());
private static final long[] mk_tokenSet_7() {
long[] data = { 143622400L, 0L};
```

```
return data;
}
public static final BitSet _tokenSet_7 = new BitSet(mk_tokenSet_7());
private static final long[] mk_tokenSet_8() {
long[] data = { 562949953421312L, 0L};
return data;
}
public static final BitSet _tokenSet_8 = new BitSet(mk_tokenSet_8());
private static final long[] mk_tokenSet_9() {
long[] data = { 1125899906842624L, 0L};
return data;
}
public static final BitSet _tokenSet_9 = new BitSet(mk_tokenSet_9());
private static final long[] mk_tokenSet_10() {
long[] data = { 123145436528928L, 0L};
return data;
}
public static final BitSet _tokenSet_10 = new BitSet(mk_tokenSet_10());
private static final long[] mk_tokenSet_11() {
long[] data = { 134217728L, 0L};
return data;
}
public static final BitSet _tokenSet_11 = new BitSet(mk_tokenSet_11());


// Marvin Rich -- Walker}
// $ANTLR 2.7.5 (20050128): “Firedrl.g” -> “FiredrlWalker.java”$

import antlr.TreeParser;
import antlr.Token;
import antlr.collections.AST;
import antlr.RecognitionException;
import antlr.ANTLRException;
import antlr.NoViableAltException;
import antlr.MismatchedTokenException;
import antlr.SemanticException;
import antlr.collections.impl.BitSet;
import antlr.ASTPair;
import antlr.collections.impl.ASTArray;

  import java.io.* ;
  import java.util.* ;
  import antlr.CommonAST;
```

```
public class FiredrlWalker extends antlr.TreeParser       implements FiredrlToken-
Types
 {

  java.util.Hashtable dict = new java.util.Hashtable();


  //-----------------------------------------
  // Lan class contains the definition of a
  // single lan segment.
  //-----------------------------------------
  public class Lan {
    String name;       // Lan Name
    String ipaddr;     // Lan IP addr in dot notation
    int    netmsk;     // Mask for Lan

    //-- Contains hosts on this lan
    Hashtable hosts = new Hashtable();
  }

  //-----------------------------------------
  // Host class contains the name, interfaces,
  // and Firewall rules for a host.
  //-----------------------------------------
  public class Host {

    String name;                //host name

    //-- Contains Lan Interfaces assocaited with this host
    java.util.Hashtable intfc = new java.util.Hashtable();

    //-- Contains Firewall rules for this host
    java.util.Hashtable rules = new java.util.Hashtable();

  }


  //-----------------------------------------
  // Intfc class defines each Lan interface to
  // a host (e.g. similar to NIC connection)
  //-----------------------------------------
  public class Intfc {
    String lname;     //Lan name
    String hname;     //Host name
    int ID;           //Host ID on Lan
  }
```

```
//----------------------------------------
// Var class for each variable used.
//----------------------------------------
public class Var {
  String name;      //variable  name
  int val;          //variable  value
}

//----------------------------------------
// Rule class contains the definition of a
// single firewall rule.
//----------------------------------------
public class Rule {
  String name;       // Rule Name
  String type;       // Rule Type (TCP,UDP or ICMP)
  String action;     // access action
  String direction;  // access direction
  String src_addr;   // src address
  String dest_addr;  // dest address
  String src_port;   // src port
  String dest_port;  // dest port
  int    icmp_type;  // icmp type
  int    tcp_con;    // tcp connect flags
}

//----------------------------------------
// TestNetWork class gets populated via
// network definition statements of language
//----------------------------------------
public class TestNetWork  {

  Hashtable  L = new Hashtable();
  Hashtable  H = new Hashtable();
  Hashtable  R = new Hashtable();
  Hashtable  V = new Hashtable();

  //-- Name existance check
  void isExist(String s) {

    if (runnable) {
      if (  !L.containsKey(s) &&
            !H.containsKey(s) &&
            !R.containsKey(s) &&
            !V.containsKey(s) ) {
        runnable = false;
```

```
      System.out.println("Error: Name:"+ s + " undefined!");
    }
  }

}

//-- Name existance check
void noExist(String s) {

  if (runnable) {
    if (  L.containsKey(s) | |
          H.containsKey(s) | |
          R.containsKey(s) | |
          V.containsKey(s) ) {
      runnable = false;
      System.out.println("Error: Name:"+ s + " already defined!");
    }
  }

}

//-- Add Lan segment definition
void addLan(Lan l) {

  //-- Check name not already defined
  noExist(l.name);

  //-- Process if no errors
  if (runnable)
    L.put(l.name,l);
}

//-- Add Host definition
void addHost(String hname) {

  //-- Check name not already defined
  noExist(hname);

  //-- Process if no errors
  if (runnable) {
   Host h = new Host();
   h.name = hname;
   H.put(hname,h);
  }
}
```

```
//-- Add Host to Lan Definition
void addHostToLan(Intfc i) {

  Lan  l = new Lan();
  Host h = new Host();

  //-- Check names exist and of proper type
  isExist(i.hname);
  isExist(i.lname);
  if (runnable) {
    if (!H.containsKey(i.hname)) {
      runnable = false;
      System.out.println("Error: Name: "+ i.hname + " not host name type!");
    }
    if (!L.containsKey(i.lname)) {
      runnable = false;
      System.out.println("Error: Name: "+ i.lname + " not lan name type!");
    }
  }

  //-- Process if no errors
  if (runnable) {

    //-- Link host to Lan
    l = (Lan) L.get(i.lname);
    l.hosts.put(i.hname,i.hname);
    L.put(l.name,l);

    //-- Add interface to host
    h = (Host) H.get(i.hname);
    h.intfc.put(i.lname,i);
    H.put(h.name,h);

  }

}



//-- Add Rule to Host Definition
void addRuleToHost(String hname, String rname) {


  //-- Check names exist and of proper types
  isExist(hname);
  isExist(rname);
```

```
  if (runnable) {
    if (!H.containsKey(hname)) {
      runnable = false;
      System.out.println("Error: Name "+ hname + " not host name type!");
    }
    if (!R.containsKey(rname)) {
      runnable = false;
      System.out.println("Error: Name "+ rname + " not rule name type!");
    }
  }

  //-- Process if no errors
  if (runnable) {
    Host h = new Host();
    h = (Host) H.get(hname);
    h.rules.put(rname,rname);
    H.put(h.name,h);
  }

}


//-- Add variable
void addVar(String vname) {

  //-- Check variable not defined already
  noExist(vname);

  //-- Process if no errors
  if (runnable) {
    Var tmp = new Var();
    tmp.name = vname;
    tmp.val  = 0;
    V.put(vname, tmp);
  }

}

//-- Assign one variable to another
void assignVar(String lname,String rname) {

  //-- Check names exist and of proper type
  isExist(lname);
  isExist(rname);
  if (runnable) {
    if (!V.containsKey(lname)) {
```

```
      runnable = false;
      System.out.println("Error: Name "+ lname + " not var type!");
    }
    if (!V.containsKey(rname)) {
      runnable = false;
      System.out.println("Error: Name "+ rname + " not var type!");
    }
  }

  //-- Process if no errors
  if (runnable) {
    Var v1 = new Var();
    Var v2 = new Var();
    v1 = (Var) V.get(rname);
    v2 = (Var) V.get(lname);
    v1.val = v2.val;
    V.put(rname,v1);
  }

}

//-- Assign constant to a variable
void assignVarConst(String vname, int val) {

  isExist(vname);
  if (runnable) {
    if (!V.containsKey(vname)) {
      runnable = false;
      System.out.println("Error: Name "+ vname + " not var type!");
    }
  }

  //-- Process if no errors
  if (runnable) {
    Var v = new Var();
    v = (Var) V.get(vname);
    v.val = val;
    V.put(vname,v);
  }

}

//-- Supports loop through Lan hosts
boolean stillHosts(Enumeration e1) {

  if (e1.hasMoreElements()) {
```

```
    cur_host = (String) e1.nextElement();
    return true;
   }
   else
    return false;

 }


 //-- Add Rule Definition
 void addRule(String rname,String b,String c, String d,
             String e, String f, String g, String h,
             int i, int j) {

   //-- Check rule name not defined already
   noExist(rname);

   //-- Process if no errors
   if (runnable) {
    Rule r  = new Rule();
    r.name= rname; r.type = b; r.action = c; r.direction = d;
    r.src_addr = e; r.dest_addr = f; r.src_port = g; r.dest_port = h;
    r.icmp_type = i; r.tcp_con = j;
    R.put(r.name,r);
   }
 }

 //-- Report Network Topology
 void reportNetwork() {

   System.out.println("+-------------------------------");
   System.out.println("+ Lan Definitions ");
   System.out.println("+-------------------------------");

   for (Enumeration e1 = L.elements();
       e1.hasMoreElements();  ) {
    Lan l = (Lan) e1.nextElement();
    System.out.println(" Lan Name = "  + l.name +
                    " IP Addr =  "  + l.ipaddr +
                    " Net Msk =  "  + l.netmsk);

    for (Enumeration e2 = l.hosts.elements();
        e2.hasMoreElements();  ) {
      String s = (String) e2.nextElement();
      System.out.println("  -- Host= " + s);
    }
```

```
    }

    System.out.println("+------------------------------");
    System.out.println("+ Host Definitions ");
    System.out.println("+------------------------------");

    for (Enumeration e1 = H.elements();
        e1.hasMoreElements();  ) {
      Host h = (Host) e1.nextElement();
      System.out.println(" Host Name = "  + h.name );

      for (Enumeration e2 = h.intfc.elements();
          e2.hasMoreElements();  ) {
        Intfc i = (Intfc) e2.nextElement();
        System.out.println(" --- Lan Name = " + i.lname);
        System.out.println("     Lan ID = " + i.ID);
      }

      for (Enumeration e2 = h.rules.elements();
          e2.hasMoreElements();  ) {
        String s = (String) e2.nextElement();
        System.out.println(" --- Rule Name = " + s);
      }
    }

    System.out.println("+------------------------------");
    System.out.println("+ Rule Definitions ");
    System.out.println("+------------------------------");
    for (Enumeration e = R.elements();
        e.hasMoreElements();  ) {
      Rule r = (Rule) e.nextElement();

      System.out.println(" Rule name = "  + r.name);
      System.out.println(" ---- type = "  + r.type +
                         " act =  "       + r.action +
                         " dir = "        + r.direction);
      System.out.println("      src =  "       + r.src_addr +
                         " s_port = "     + r.src_port +
                         " dest = "       + r.dest_addr +
                         " d_port = "     + r.dest_port +
                         " i_type = "     + r.icmp_type +
                         " con ="         + r.tcp_con );
    }
  }
}
```

```
  boolean runnable = true;              //False if semantic errors found
  String cur_host = "C";                //Current host name within loop
  TestNetWork ntwk = new TestNetWork();  //Create Test Network Environment

public FiredrlWalker() {
tokenNames = _tokenNames;
}

public final void file(AST _t) throws RecognitionException {

AST file_AST_in = (_t == ASTNULL) ? null : (AST)_t;
returnAST = null;
ASTPair currentAST = new ASTPair();
AST file_AST = null;
int r;

try {      // for error handling
{
int _cnt89=0;
_loop89:
do {
if (_t==null) _t=ASTNULL;
if ((_tokenSet_0.member(_t.getType()))) {
r=stmt(_t);
_t = _retTree;
astFactory.addASTChild(currentAST, returnAST);
}
else {
if ( _cnt89>=1 ) { break _loop89; } else {throw new NoViableAltException(_t);}
}

_cnt89++;
} while (true);
}
file_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
if (_t!=null) {_t = _t.getNextSibling();}
}
returnAST = file_AST;
_retTree = _t;
}

public final int  stmt(AST _t) throws RecognitionException {
int r=0;;
```

```
AST stmt_AST_in = (_t == ASTNULL) ? null : (AST)_t;
returnAST = null;
ASTPair currentAST = new ASTPair();
AST stmt_AST = null;
AST d = null;
AST d_AST = null;
AST p_AST = null;
AST p = null;
AST a = null;
AST a_AST = null;
AST n1 = null;
AST n1_AST = null;
AST c1 = null;
AST c1_AST = null;
AST n2 = null;
AST n2_AST = null;
AST n3 = null;
AST n3_AST = null;
AST c2 = null;
AST c2_AST = null;

Lan    l = new Lan();
Host   h = new Host();
Intfc  i = new Intfc();
String s1,s2,p1,p2;
String act,dir,name;
Intfc  i2;


try {      // for error handling
if (_t==null) _t=ASTNULL;
switch ( _t.getType()) {
case LITERAL_lan_def:
{
AST __t91 = _t;
AST tmp1_AST = null;
AST tmp1_AST_in = null;
tmp1_AST = astFactory.create((AST)_t);
tmp1_AST_in = (AST)_t;
astFactory.addASTChild(currentAST, tmp1_AST);
ASTPair __currentAST91 = currentAST.copy();
currentAST.root = currentAST.child;
currentAST.child = null;
match(_t,LITERAL_lan_def);
_t = _t.getFirstChild();
```

```
{
int _cnt93=0;
_loop93:
do {
if (_t==null) _t=ASTNULL;
if ((_t.getType()==COLON)) {
l=lan(_t);
_t = _retTree;
astFactory.addASTChild(currentAST, returnAST);
ntwk.addLan(l); r = 0;
}
else {
if ( _cnt93>=1 ) { break _loop93; } else {throw new NoViableAltException(_t);}
}

_cnt93++;
} while (true);
}
currentAST = __currentAST91;
_t = __t91;
_t = _t.getNextSibling();
stmt_AST = (AST)currentAST.root;
break;
}
case LITERAL_host_def:
{
AST __t94 = _t;
AST tmp2_AST = null;
AST tmp2_AST_in = null;
tmp2_AST = astFactory.create((AST)_t);
tmp2_AST_in = (AST)_t;
astFactory.addASTChild(currentAST, tmp2_AST);
ASTPair __currentAST94 = currentAST.copy();
currentAST.root = currentAST.child;
currentAST.child = null;
match(_t,LITERAL_host_def);
_t = _t.getFirstChild();
{
int _cnt96=0;
_loop96:
do {
if (_t==null) _t=ASTNULL;
if ((_t.getType()==ID)) {
AST tmp3_AST = null;
AST tmp3_AST_in = null;
tmp3_AST = astFactory.create((AST)_t);
```

```
tmp3_AST_in = (AST)_t;
astFactory.addASTChild(currentAST, tmp3_AST);
match(_t,ID);
_t = _t.getNextSibling();
ntwk.addHost(tmp3_AST.getText()); r = 0;
}
else {
if ( _cnt96>=1 ) { break _loop96; } else {throw new NoViableAltException(_t);}
}

_cnt96++;
} while (true);
}
currentAST = __currentAST94;
_t = __t94;
_t = _t.getNextSibling();
stmt_AST = (AST)currentAST.root;
break;
}
case LITERAL_var:
{
AST __t97 = _t;
AST tmp4_AST = null;
AST tmp4_AST_in = null;
tmp4_AST = astFactory.create((AST)_t);
tmp4_AST_in = (AST)_t;
astFactory.addASTChild(currentAST, tmp4_AST);
ASTPair __currentAST97 = currentAST.copy();
currentAST.root = currentAST.child;
currentAST.child = null;
match(_t,LITERAL_var);
_t = _t.getFirstChild();
{
int _cnt99=0;
_loop99:
do {
if (_t==null) _t=ASTNULL;
if ((_t.getType()==ID)) {
AST tmp5_AST = null;
AST tmp5_AST_in = null;
tmp5_AST = astFactory.create((AST)_t);
tmp5_AST_in = (AST)_t;
astFactory.addASTChild(currentAST, tmp5_AST);
match(_t,ID);
_t = _t.getNextSibling();
ntwk.addVar(tmp5_AST.getText()); r = 0;
```

```
}
else {
if ( _cnt99>=1 ) { break _loop99; } else {throw new NoViableAltException(_t);}
}

_cnt99++;
} while (true);
}
currentAST = __currentAST97;
_t = __t97;
_t = _t.getNextSibling();
stmt_AST = (AST)currentAST.root;
break;
}
case 40:
{
AST __t100 = _t;
AST tmp6_AST = null;
AST tmp6_AST_in = null;
tmp6_AST = astFactory.create((AST)_t);
tmp6_AST_in = (AST)_t;
astFactory.addASTChild(currentAST, tmp6_AST);
ASTPair __currentAST100 = currentAST.copy();
currentAST.root = currentAST.child;
currentAST.child = null;
match(_t,40);
_t = _t.getFirstChild();
AST tmp7_AST = null;
AST tmp7_AST_in = null;
tmp7_AST = astFactory.create((AST)_t);
tmp7_AST_in = (AST)_t;
astFactory.addASTChild(currentAST, tmp7_AST);
match(_t,ID);
_t = _t.getNextSibling();
i.lname = tmp7_AST.getText();
{
int _cnt102=0;
_loop102:
do {
if (_t==null) _t=ASTNULL;
if ((_t.getType()==ID)) {
i2=intfc(_t,i);
_t = _retTree;
astFactory.addASTChild(currentAST, returnAST);
ntwk.addHostToLan(i2); r = 0;
}
```

```
else {
if ( _cnt102>=1 ) { break _loop102; } else {throw new NoViableAltException(_t);}
}

_cnt102++;
} while (true);
}
currentAST = __currentAST100;
_t = __t100;
_t = _t.getNextSibling();
stmt_AST = (AST)currentAST.root;
break;
}
case 41:
{
AST __t103 = _t;
AST tmp8_AST = null;
AST tmp8_AST_in = null;
tmp8_AST = astFactory.create((AST)_t);
tmp8_AST_in = (AST)_t;
astFactory.addASTChild(currentAST, tmp8_AST);
ASTPair __currentAST103 = currentAST.copy();
currentAST.root = currentAST.child;
currentAST.child = null;
match(_t,41);
_t = _t.getFirstChild();
d = (AST)_t;
AST d_AST_in = null;
d_AST = astFactory.create(d);
astFactory.addASTChild(currentAST, d_AST);
match(_t,ID);
_t = _t.getNextSibling();
{
int _cnt105=0;
_loop105:
do {
if (_t==null) _t=ASTNULL;
if ((_t.getType()==ID)) {
AST tmp9_AST = null;
AST tmp9_AST_in = null;
tmp9_AST = astFactory.create((AST)_t);
tmp9_AST_in = (AST)_t;
astFactory.addASTChild(currentAST, tmp9_AST);
match(_t,ID);
_t = _t.getNextSibling();
ntwk.addRuleToHost(d.getText(),tmp9_AST.getText()); r=0;
```

```
}
else {
if ( _cnt105>=1 ) { break _loop105; } else {throw new NoViableAltException(_t);}
}

_cnt105++;
} while (true);
}
currentAST = __currentAST103;
_t = __t103;
_t = _t.getNextSibling();
stmt_AST = (AST)currentAST.root;
break;
}
case 42:
{
AST __t106 = _t;
AST tmp10_AST = null;
AST tmp10_AST_in = null;
tmp10_AST = astFactory.create((AST)_t);
tmp10_AST_in = (AST)_t;
astFactory.addASTChild(currentAST, tmp10_AST);
ASTPair __currentAST106 = currentAST.copy();
currentAST.root = currentAST.child;
currentAST.child = null;
match(_t,42);
_t = _t.getFirstChild();
{
int _cnt108=0;
_loop108:
do {
if (_t==null) _t=ASTNULL;
if ((_t.getType()==ID)) {
AST tmp11_AST = null;
AST tmp11_AST_in = null;
tmp11_AST = astFactory.create((AST)_t);
tmp11_AST_in = (AST)_t;
astFactory.addASTChild(currentAST, tmp11_AST);
match(_t,ID);
_t = _t.getNextSibling();
ntwk.addRuleToHost(cur_host,tmp11_AST.getText()); r=0;
}
else {
if ( _cnt108>=1 ) { break _loop108; } else {throw new NoViableAltException(_t);}
}
```

```
_cnt108++;
} while (true);
}
currentAST = __currentAST106;
_t = __t106;
_t = _t.getNextSibling();
stmt_AST = (AST)currentAST.root;
break;
}
case LITERAL_if:
{
AST __t109 = _t;
AST tmp12_AST = null;
AST tmp12_AST_in = null;
tmp12_AST = astFactory.create((AST)_t);
tmp12_AST_in = (AST)_t;
astFactory.addASTChild(currentAST, tmp12_AST);
ASTPair __currentAST109 = currentAST.copy();
currentAST.root = currentAST.child;
currentAST.child = null;
match(_t,LITERAL_if);
_t = _t.getFirstChild();
p = _t==ASTNULL ? null : (AST)_t;
s1=pred(_t);
_t = _retTree;
p_AST = (AST)returnAST;
astFactory.addASTChild(currentAST, returnAST);

AST thenpart = p.getNextSibling();
System.out.println("At if statement thenpart");
if (true) r=stmt(thenpart);
r=0;

currentAST = __currentAST109;
_t = __t109;
_t = _t.getNextSibling();
stmt_AST = (AST)currentAST.root;
break;
}
case LITERAL_for:
{
AST __t110 = _t;
AST tmp13_AST = null;
AST tmp13_AST_in = null;
tmp13_AST = astFactory.create((AST)_t);
tmp13_AST_in = (AST)_t;
```

```
astFactory.addASTChild(currentAST, tmp13_AST);
ASTPair __currentAST110 = currentAST.copy();
currentAST.root = currentAST.child;
currentAST.child = null;
match(_t,LITERAL_for);
_t = _t.getFirstChild();
a = (AST)_t;
AST a_AST_in = null;
a_AST = astFactory.create(a);
astFactory.addASTChild(currentAST, a_AST);
match(_t,ID);
_t = _t.getNextSibling();

l = (Lan) ntwk.L.get(a.getText());
Enumeration e1 = l.hosts.elements();
while (ntwk.stillHosts(e1)) {
AST strt_of_block = a.getNextSibling();
stmt(strt_of_block);
}

currentAST = __currentAST110;
_t = __t110;
_t = _t.getNextSibling();
stmt_AST = (AST)currentAST.root;
break;
}
case RBRKT:
{
AST __t111 = _t;
AST tmp14_AST = null;
AST tmp14_AST_in = null;
tmp14_AST = astFactory.create((AST)_t);
tmp14_AST_in = (AST)_t;
astFactory.addASTChild(currentAST, tmp14_AST);
ASTPair __currentAST111 = currentAST.copy();
currentAST.root = currentAST.child;
currentAST.child = null;
match(_t,RBRKT);
_t = _t.getFirstChild();

l = (Lan) ntwk.L.get(a.getText());
Enumeration e1 = l.hosts.elements();
while (ntwk.stillHosts(e1)) {
AST strt_of_block = a.getNextSibling();
stmt(strt_of_block);
}
```

```
currentAST = __currentAST111;
_t = __t111;
_t = _t.getNextSibling();
stmt_AST = (AST)currentAST.root;
break;
}
case ASSIGN:
{
AST __t112 = _t;
AST tmp15_AST = null;
AST tmp15_AST_in = null;
tmp15_AST = astFactory.create((AST)_t);
tmp15_AST_in = (AST)_t;
astFactory.addASTChild(currentAST, tmp15_AST);
ASTPair __currentAST112 = currentAST.copy();
currentAST.root = currentAST.child;
currentAST.child = null;
match(_t,ASSIGN);
_t = _t.getFirstChild();
AST tmp16_AST = null;
AST tmp16_AST_in = null;
tmp16_AST = astFactory.create((AST)_t);
tmp16_AST_in = (AST)_t;
astFactory.addASTChild(currentAST, tmp16_AST);
match(_t,ID);
_t = _t.getNextSibling();
var_src(_t,tmp16_AST.getText());
_t = _retTree;
astFactory.addASTChild(currentAST, returnAST);
r = 0;
currentAST = __currentAST112;
_t = __t112;
_t = _t.getNextSibling();
stmt_AST = (AST)currentAST.root;
break;
}
case LITERAL_TCP:
{
AST __t113 = _t;
AST tmp17_AST = null;
AST tmp17_AST_in = null;
tmp17_AST = astFactory.create((AST)_t);
tmp17_AST_in = (AST)_t;
astFactory.addASTChild(currentAST, tmp17_AST);
ASTPair __currentAST113 = currentAST.copy();
```

```
currentAST.root = currentAST.child;
currentAST.child = null;
match(_t,LITERAL_TCP);
_t = _t.getFirstChild();
n1 = (AST)_t;
AST n1_AST_in = null;
n1_AST = astFactory.create(n1);
astFactory.addASTChild(currentAST, n1_AST);
match(_t,ID);
_t = _t.getNextSibling();
act=action(_t);
_t = _retTree;
astFactory.addASTChild(currentAST, returnAST);
dir=direction(_t);
_t = _retTree;
astFactory.addASTChild(currentAST, returnAST);
s1=ip_sel(_t);
_t = _retTree;
astFactory.addASTChild(currentAST, returnAST);
s2=ip_sel(_t);
_t = _retTree;
astFactory.addASTChild(currentAST, returnAST);
p1=port(_t);
_t = _retTree;
astFactory.addASTChild(currentAST, returnAST);
p2=port(_t);
_t = _retTree;
astFactory.addASTChild(currentAST, returnAST);
c1 = (AST)_t;
AST c1_AST_in = null;
c1_AST = astFactory.create(c1);
astFactory.addASTChild(currentAST, c1_AST);
match(_t,INT);
_t = _t.getNextSibling();
ntwk.addRule(n1.getText(),"TCP",act,dir,s1,s2,p1,p2,
0,Integer.parseInt(c1.getText())); r=0;
currentAST = __currentAST113;
_t = __t113;
_t = _t.getNextSibling();
stmt_AST = (AST)currentAST.root;
break;
}
case LITERAL_UDP:
{
AST __t114 = _t;
AST tmp18_AST = null;
```

```
AST tmp18_AST_in = null;
tmp18_AST = astFactory.create((AST)_t);
tmp18_AST_in = (AST)_t;
astFactory.addASTChild(currentAST, tmp18_AST);
ASTPair __currentAST114 = currentAST.copy();
currentAST.root = currentAST.child;
currentAST.child = null;
match(_t,LITERAL_UDP);
_t = _t.getFirstChild();
n2 = (AST)_t;
AST n2_AST_in = null;
n2_AST = astFactory.create(n2);
astFactory.addASTChild(currentAST, n2_AST);
match(_t,ID);
_t = _t.getNextSibling();
act=action(_t);
_t = _retTree;
astFactory.addASTChild(currentAST, returnAST);
dir=direction(_t);
_t = _retTree;
astFactory.addASTChild(currentAST, returnAST);
s1=ip_sel(_t);
_t = _retTree;
astFactory.addASTChild(currentAST, returnAST);
s2=ip_sel(_t);
_t = _retTree;
astFactory.addASTChild(currentAST, returnAST);
p1=port(_t);
_t = _retTree;
astFactory.addASTChild(currentAST, returnAST);
p2=port(_t);
_t = _retTree;
astFactory.addASTChild(currentAST, returnAST);
ntwk.addRule(n2.getText(),"UDP",act,dir,s1,s2,p1,p2,0,0); r=0;
currentAST = __currentAST114;
_t = __t114;
_t = _t.getNextSibling();
stmt_AST = (AST)currentAST.root;
break;
}
case LITERAL_ICMP:
{
AST __t115 = _t;
AST tmp19_AST = null;
AST tmp19_AST_in = null;
tmp19_AST = astFactory.create((AST)_t);
```

```
tmp19_AST_in = (AST)_t;
astFactory.addASTChild(currentAST, tmp19_AST);
ASTPair __currentAST115 = currentAST.copy();
currentAST.root = currentAST.child;
currentAST.child = null;
match(_t,LITERAL_ICMP);
_t = _t.getFirstChild();
n3 = (AST)_t;
AST n3_AST_in = null;
n3_AST = astFactory.create(n3);
astFactory.addASTChild(currentAST, n3_AST);
match(_t,ID);
_t = _t.getNextSibling();
act=action(_t);
_t = _retTree;
astFactory.addASTChild(currentAST, returnAST);
dir=direction(_t);
_t = _retTree;
astFactory.addASTChild(currentAST, returnAST);
s1=ip_sel(_t);
_t = _retTree;
astFactory.addASTChild(currentAST, returnAST);
s2=ip_sel(_t);
_t = _retTree;
astFactory.addASTChild(currentAST, returnAST);
c2 = (AST)_t;
AST c2_AST_in = null;
c2_AST = astFactory.create(c2);
astFactory.addASTChild(currentAST, c2_AST);
match(_t,INT);
_t = _t.getNextSibling();
ntwk.addRule(n3.getText(),"ICMP",act,dir,s1,s2,"","",
Integer.parseInt(c2.getText()),0); r=0;
currentAST = __currentAST115;
_t = __t115;
_t = _t.getNextSibling();
stmt_AST = (AST)currentAST.root;
break;
}
case LITERAL_report:
{
AST __t116 = _t;
AST tmp20_AST = null;
AST tmp20_AST_in = null;
tmp20_AST = astFactory.create((AST)_t);
tmp20_AST_in = (AST)_t;
```

```
astFactory.addASTChild(currentAST, tmp20_AST);
ASTPair __currentAST116 = currentAST.copy();
currentAST.root = currentAST.child;
currentAST.child = null;
match(_t,LITERAL_report);
_t = _t.getFirstChild();
s1=rpt_targ(_t);
_t = _retTree;
astFactory.addASTChild(currentAST, returnAST);
p1=rpt_proto(_t);
_t = _retTree;
astFactory.addASTChild(currentAST, returnAST);
s2=rpt_dir(_t);
_t = _retTree;
astFactory.addASTChild(currentAST, returnAST);
r=0;
currentAST = __currentAST116;
_t = __t116;
_t = _t.getNextSibling();
stmt_AST = (AST)currentAST.root;
break;
}
case LITERAL_topology:
{
AST __t117 = _t;
AST tmp21_AST = null;
AST tmp21_AST_in = null;
tmp21_AST = astFactory.create((AST)_t);
tmp21_AST_in = (AST)_t;
astFactory.addASTChild(currentAST, tmp21_AST);
ASTPair __currentAST117 = currentAST.copy();
currentAST.root = currentAST.child;
currentAST.child = null;
match(_t,LITERAL_topology);
_t = _t.getFirstChild();
ntwk.reportNetwork(); r=0;
currentAST = __currentAST117;
_t = __t117;
_t = _t.getNextSibling();
stmt_AST = (AST)currentAST.root;
break;
}
default:
{
throw new NoViableAltException(_t);
}
```

```
}
}
catch (RecognitionException ex) {
reportError(ex);
if (_t!=null) {_t = _t.getNextSibling();}
}
returnAST = stmt_AST;
_retTree = _t;
return r;
}

public final Lan  lan(AST _t) throws RecognitionException {
Lan lan = new Lan();;

AST lan_AST_in = (_t == ASTNULL) ? null : (AST)_t;
returnAST = null;
ASTPair currentAST = new ASTPair();
AST lan_AST = null;
AST a = null;
AST a_AST = null;
AST b = null;
AST b_AST = null;
String s="";

try {      // for error handling
AST __t121 = _t;
AST tmp22_AST = null;
AST tmp22_AST_in = null;
tmp22_AST = astFactory.create((AST)_t);
tmp22_AST_in = (AST)_t;
astFactory.addASTChild(currentAST, tmp22_AST);
ASTPair __currentAST121 = currentAST.copy();
currentAST.root = currentAST.child;
currentAST.child = null;
match(_t,COLON);
_t = _t.getFirstChild();
a = (AST)_t;
AST a_AST_in = null;
a_AST = astFactory.create(a);
astFactory.addASTChild(currentAST, a_AST);
match(_t,ID);
_t = _t.getNextSibling();
lan.name = a.getText();
s=ipaddr(_t);
_t = _retTree;
astFactory.addASTChild(currentAST, returnAST);
```

```
lan.ipaddr = s;
b = (AST)_t;
AST b_AST_in = null;
b_AST = astFactory.create(b);
astFactory.addASTChild(currentAST, b_AST);
match(_t,INT);
_t = _t.getNextSibling();
lan.netmsk = Integer.parseInt(b.getText());
currentAST = __currentAST121;
_t = __t121;
_t = _t.getNextSibling();
lan_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
if (_t!=null) {_t = _t.getNextSibling();}
}
returnAST = lan_AST;
_retTree = _t;
return lan;
}

public final Intfc  intfc(AST _t,
Intfc i2
) throws RecognitionException {
Intfc i = new Intfc();;

AST intfc_AST_in = (_t == ASTNULL) ? null : (AST)_t;
returnAST = null;
ASTPair currentAST = new ASTPair();
AST intfc_AST = null;
AST a = null;
AST a_AST = null;
i.lname = i2.lname;

try {      // for error handling
AST tmp23_AST = null;
AST tmp23_AST_in = null;
tmp23_AST = astFactory.create((AST)_t);
tmp23_AST_in = (AST)_t;
astFactory.addASTChild(currentAST, tmp23_AST);
match(_t,ID);
_t = _t.getNextSibling();
i.hname = tmp23_AST.getText();
a = (AST)_t;
AST a_AST_in = null;
```

```
a_AST = astFactory.create(a);
astFactory.addASTChild(currentAST, a_AST);
match(_t,INT);
_t = _t.getNextSibling();
i.ID = Integer.parseInt(a.getText());
intfc_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
if (_t!=null) {_t = _t.getNextSibling();}
}
returnAST = intfc_AST;
_retTree = _t;
return i;
}

public final String  pred(AST _t) throws RecognitionException {
String s="";;

AST pred_AST_in = (_t == ASTNULL) ? null : (AST)_t;
returnAST = null;
ASTPair currentAST = new ASTPair();
AST pred_AST = null;
AST b = null;
AST b_AST = null;

try {      // for error handling
AST __t119 = _t;
AST tmp24_AST = null;
AST tmp24_AST_in = null;
tmp24_AST = astFactory.create((AST)_t);
tmp24_AST_in = (AST)_t;
astFactory.addASTChild(currentAST, tmp24_AST);
ASTPair __currentAST119 = currentAST.copy();
currentAST.root = currentAST.child;
currentAST.child = null;
match(_t,57);
_t = _t.getFirstChild();
AST tmp25_AST = null;
AST tmp25_AST_in = null;
tmp25_AST = astFactory.create((AST)_t);
tmp25_AST_in = (AST)_t;
astFactory.addASTChild(currentAST, tmp25_AST);
match(_t,ID);
_t = _t.getNextSibling();
s = tmp25_AST.getText();
```

```
s += ">";
b = (AST)_t;
AST b_AST_in = null;
b_AST = astFactory.create(b);
astFactory.addASTChild(currentAST, b_AST);
match(_t,INT);
_t = _t.getNextSibling();
s += b;
currentAST = __currentAST119;
_t = __t119;
_t = _t.getNextSibling();
pred_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
if (_t!=null) {_t = _t.getNextSibling();}
}
returnAST = pred_AST;
_retTree = _t;
return s;
}

public final void var_src(AST _t,
String s
) throws RecognitionException {

AST var_src_AST_in = (_t == ASTNULL) ? null : (AST)_t;
returnAST = null;
ASTPair currentAST = new ASTPair();
AST var_src_AST = null;
AST b = null;
AST b_AST = null;

try {      // for error handling
{
if (_t==null) _t=ASTNULL;
switch ( _t.getType()) {
case ID:
{
{
AST tmp26_AST = null;
AST tmp26_AST_in = null;
tmp26_AST = astFactory.create((AST)_t);
tmp26_AST_in = (AST)_t;
astFactory.addASTChild(currentAST, tmp26_AST);
match(_t,ID);
```

```
_t = _t.getNextSibling();
ntwk.assignVar(s,tmp26_AST.getText());
}
break;
}
case INT:
{
{
b = (AST)_t;
AST b_AST_in = null;
b_AST = astFactory.create(b);
astFactory.addASTChild(currentAST, b_AST);
match(_t,INT);
_t = _t.getNextSibling();
ntwk.assignVarConst(s,Integer.parseInt(b.getText()));
}
break;
}
default:
{
throw new NoViableAltException(_t);
}
}
}
var_src_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
if (_t!=null) {_t = _t.getNextSibling();}
}
returnAST = var_src_AST;
_retTree = _t;
}

public final String  action(AST _t) throws RecognitionException {
String s = ““;;

AST action_AST_in = (_t == ASTNULL) ? null : (AST)_t;
returnAST = null;
ASTPair currentAST = new ASTPair();
AST action_AST = null;
AST a = null;
AST a_AST = null;
AST b = null;
AST b_AST = null;
```

```
try {      // for error handling
{
if (_t==null) _t=ASTNULL;
switch ( _t.getType()) {
case LITERAL_allow:
{
{
a = (AST)_t;
AST a_AST_in = null;
a_AST = astFactory.create(a);
astFactory.addASTChild(currentAST, a_AST);
match(_t,LITERAL_allow);
_t = _t.getNextSibling();
s += a;
}
break;
}
case LITERAL_block:
{
{
b = (AST)_t;
AST b_AST_in = null;
b_AST = astFactory.create(b);
astFactory.addASTChild(currentAST, b_AST);
match(_t,LITERAL_block);
_t = _t.getNextSibling();
s += b;
}
break;
}
default:
{
throw new NoViableAltException(_t);
}
}
}
action_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
if (_t!=null) {_t = _t.getNextSibling();}
}
returnAST = action_AST;
_retTree = _t;
return s;
}
```

```
public final String  direction(AST _t) throws RecognitionException {
String s = ““;;

AST direction_AST_in = (_t == ASTNULL) ? null : (AST)_t;
returnAST = null;
ASTPair currentAST = new ASTPair();
AST direction_AST = null;
AST a = null;
AST a_AST = null;
AST b = null;
AST b_AST = null;

try {      // for error handling
{
if (_t==null) _t=ASTNULL;
switch ( _t.getType()) {
case LITERAL_in:
{
{
a = (AST)_t;
AST a_AST_in = null;
a_AST = astFactory.create(a);
astFactory.addASTChild(currentAST, a_AST);
match(_t,LITERAL_in);
_t = _t.getNextSibling();
s += a;
}
break;
}
case LITERAL_out:
{
{
b = (AST)_t;
AST b_AST_in = null;
b_AST = astFactory.create(b);
astFactory.addASTChild(currentAST, b_AST);
match(_t,LITERAL_out);
_t = _t.getNextSibling();
s += b;
}
break;
}
default:
{
throw new NoViableAltException(_t);
```

```
}
}
}
direction_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
if (_t!=null) {_t = _t.getNextSibling();}
}
returnAST = direction_AST;
_retTree = _t;
return s;
}

public final String  ip_sel(AST _t) throws RecognitionException {
String s = ““;;

AST ip_sel_AST_in = (_t == ASTNULL) ? null : (AST)_t;
returnAST = null;
ASTPair currentAST = new ASTPair();
AST ip_sel_AST = null;
AST b = null;
AST b_AST = null;
String a;

try {      // for error handling
{
if (_t==null) _t=ASTNULL;
switch ( _t.getType()) {
case DOT:
{
{
a=ipaddr(_t);
_t = _retTree;
astFactory.addASTChild(currentAST, returnAST);
s += a;
}
break;
}
case STAR:
{
{
b = (AST)_t;
AST b_AST_in = null;
b_AST = astFactory.create(b);
astFactory.addASTChild(currentAST, b_AST);
```

```
match(_t,STAR);
_t = _t.getNextSibling();
s += b;
}
break;
}
default:
{
throw new NoViableAltException(_t);
}
}
}
ip_sel_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
if (_t!=null) {_t = _t.getNextSibling();}
}
returnAST = ip_sel_AST;
_retTree = _t;
return s;
}

public final String  port(AST _t) throws RecognitionException {
String s = ““;;

AST port_AST_in = (_t == ASTNULL) ? null : (AST)_t;
returnAST = null;
ASTPair currentAST = new ASTPair();
AST port_AST = null;
AST b = null;
AST b_AST = null;
AST c = null;
AST c_AST = null;
String a;

try {      // for error handling
{
if (_t==null) _t=ASTNULL;
switch ( _t.getType()) {
case EQ:
case LE:
case GE:
case GT:
case LT:
case INT:
```

```
{
{
if (_t==null) _t=ASTNULL;
switch ( _t.getType()) {
case EQ:
case LE:
case GE:
case GT:
case LT:
{
a=cmpr(_t);
_t = _retTree;
astFactory.addASTChild(currentAST, returnAST);
s += a;
break;
}
case INT:
{
break;
}
default:
{
throw new NoViableAltException(_t);
}
}
}
b = (AST)_t;
AST b_AST_in = null;
b_AST = astFactory.create(b);
astFactory.addASTChild(currentAST, b_AST);
match(_t,INT);
_t = _t.getNextSibling();
s += b;
break;
}
case STAR:
{
c = (AST)_t;
AST c_AST_in = null;
c_AST = astFactory.create(c);
astFactory.addASTChild(currentAST, c_AST);
match(_t,STAR);
_t = _t.getNextSibling();
s += c;
break;
}
```

```
default:
{
throw new NoViableAltException(_t);
}
}
}
port_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
if (_t!=null) {_t = _t.getNextSibling();}
}
returnAST = port_AST;
_retTree = _t;
return s;
}

public final String  rpt_targ(AST _t) throws RecognitionException {
String s = ““;;

AST rpt_targ_AST_in = (_t == ASTNULL) ? null : (AST)_t;
returnAST = null;
ASTPair currentAST = new ASTPair();
AST rpt_targ_AST = null;
AST a = null;
AST a_AST = null;
AST b = null;
AST b_AST = null;

try {      // for error handling
{
if (_t==null) _t=ASTNULL;
switch ( _t.getType()) {
case ID:
{
{
a = (AST)_t;
AST a_AST_in = null;
a_AST = astFactory.create(a);
astFactory.addASTChild(currentAST, a_AST);
match(_t,ID);
_t = _t.getNextSibling();
s += a.getText();
}
break;
}
```

```
case STAR:
{
{
b = (AST)_t;
AST b_AST_in = null;
b_AST = astFactory.create(b);
astFactory.addASTChild(currentAST, b_AST);
match(_t,STAR);
_t = _t.getNextSibling();
s += b;
}
break;
}
default:
{
throw new NoViableAltException(_t);
}
}
}
rpt_targ_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
if (_t!=null) {_t = _t.getNextSibling();}
}
returnAST = rpt_targ_AST;
_retTree = _t;
return s;
}

public final String  rpt_proto(AST _t) throws RecognitionException {
String s = ““;;

AST rpt_proto_AST_in = (_t == ASTNULL) ? null : (AST)_t;
returnAST = null;
ASTPair currentAST = new ASTPair();
AST rpt_proto_AST = null;
AST a = null;
AST a_AST = null;
AST b = null;
AST b_AST = null;
AST c = null;
AST c_AST = null;
AST d = null;
AST d_AST = null;
```

```
try {      // for error handling
{
if (_t==null) _t=ASTNULL;
switch ( _t.getType()) {
case LITERAL_TCP:
{
{
a = (AST)_t;
AST a_AST_in = null;
a_AST = astFactory.create(a);
astFactory.addASTChild(currentAST, a_AST);
match(_t,LITERAL_TCP);
_t = _t.getNextSibling();
s += a;
}
break;
}
case LITERAL_UDP:
{
{
b = (AST)_t;
AST b_AST_in = null;
b_AST = astFactory.create(b);
astFactory.addASTChild(currentAST, b_AST);
match(_t,LITERAL_UDP);
_t = _t.getNextSibling();
s += b;
}
break;
}
case LITERAL_ICMP:
{
{
c = (AST)_t;
AST c_AST_in = null;
c_AST = astFactory.create(c);
astFactory.addASTChild(currentAST, c_AST);
match(_t,LITERAL_ICMP);
_t = _t.getNextSibling();
s += c;
}
break;
}
case STAR:
{
{
```

```
d = (AST)_t;
AST d_AST_in = null;
d_AST = astFactory.create(d);
astFactory.addASTChild(currentAST, d_AST);
match(_t,STAR);
_t = _t.getNextSibling();
s += d ;
}
break;
}
default:
{
throw new NoViableAltException(_t);
}
}
}
rpt_proto_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
if (_t!=null) {_t = _t.getNextSibling();}
}
returnAST = rpt_proto_AST;
_retTree = _t;
return s;
}

public final String  rpt_dir(AST _t) throws RecognitionException {
String s = ““;;

AST rpt_dir_AST_in = (_t == ASTNULL) ? null : (AST)_t;
returnAST = null;
ASTPair currentAST = new ASTPair();
AST rpt_dir_AST = null;
AST a = null;
AST a_AST = null;
AST b = null;
AST b_AST = null;
AST c = null;
AST c_AST = null;

try {      // for error handling
{
if (_t==null) _t=ASTNULL;
switch ( _t.getType()) {
case LITERAL_snd:
```

```
{
{
a = (AST)_t;
AST a_AST_in = null;
a_AST = astFactory.create(a);
astFactory.addASTChild(currentAST, a_AST);
match(_t,LITERAL_snd);
_t = _t.getNextSibling();
s += a;
}
break;
}
case LITERAL_rcv:
{
{
b = (AST)_t;
AST b_AST_in = null;
b_AST = astFactory.create(b);
astFactory.addASTChild(currentAST, b_AST);
match(_t,LITERAL_rcv);
_t = _t.getNextSibling();
s += b;
}
break;
}
case STAR:
{
{
c = (AST)_t;
AST c_AST_in = null;
c_AST = astFactory.create(c);
astFactory.addASTChild(currentAST, c_AST);
match(_t,STAR);
_t = _t.getNextSibling();
s += c;
}
break;
}
default:
{
throw new NoViableAltException(_t);
}
}
}
rpt_dir_AST = (AST)currentAST.root;
}
```

```
catch (RecognitionException ex) {
reportError(ex);
if (_t!=null) {_t = _t.getNextSibling();}
}
returnAST = rpt_dir_AST;
_retTree = _t;
return s;
}

public final String  ipaddr(AST _t) throws RecognitionException {
String s = ““;;

AST ipaddr_AST_in = (_t == ASTNULL) ? null : (AST)_t;
returnAST = null;
ASTPair currentAST = new ASTPair();
AST ipaddr_AST = null;
AST a = null;
AST a_AST = null;
AST b = null;
AST b_AST = null;
AST c = null;
AST c_AST = null;
AST d = null;
AST d_AST = null;

try {      // for error handling
AST __t123 = _t;
AST tmp27_AST = null;
AST tmp27_AST_in = null;
tmp27_AST = astFactory.create((AST)_t);
tmp27_AST_in = (AST)_t;
astFactory.addASTChild(currentAST, tmp27_AST);
ASTPair __currentAST123 = currentAST.copy();
currentAST.root = currentAST.child;
currentAST.child = null;
match(_t,DOT);
_t = _t.getFirstChild();
a = (AST)_t;
AST a_AST_in = null;
a_AST = astFactory.create(a);
astFactory.addASTChild(currentAST, a_AST);
match(_t,INT);
_t = _t.getNextSibling();
b = (AST)_t;
AST b_AST_in = null;
b_AST = astFactory.create(b);
```

```
astFactory.addASTChild(currentAST, b_AST);
match(_t,INT);
_t = _t.getNextSibling();
c = (AST)_t;
AST c_AST_in = null;
c_AST = astFactory.create(c);
astFactory.addASTChild(currentAST, c_AST);
match(_t,INT);
_t = _t.getNextSibling();
d = (AST)_t;
AST d_AST_in = null;
d_AST = astFactory.create(d);
astFactory.addASTChild(currentAST, d_AST);
match(_t,INT);
_t = _t.getNextSibling();
s = a+"."+b+"."+c+"."+d;
currentAST = __currentAST123;
_t = __t123;
_t = _t.getNextSibling();
ipaddr_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
if (_t!=null) {_t = _t.getNextSibling();}
}
returnAST = ipaddr_AST;
_retTree = _t;
return s;
}

public final String  cmpr(AST _t) throws RecognitionException {
String s = ““;;

AST cmpr_AST_in = (_t == ASTNULL) ? null : (AST)_t;
returnAST = null;
ASTPair currentAST = new ASTPair();
AST cmpr_AST = null;
AST a = null;
AST a_AST = null;
AST b = null;
AST b_AST = null;
AST c = null;
AST c_AST = null;
AST d = null;
AST d_AST = null;
AST e = null;
```

```
AST e_AST = null;

try {      // for error handling
if (_t==null) _t=ASTNULL;
switch ( _t.getType()) {
case GT:
{
{
a = (AST)_t;
AST a_AST_in = null;
a_AST = astFactory.create(a);
astFactory.addASTChild(currentAST, a_AST);
match(_t,GT);
_t = _t.getNextSibling();
s += a;
}
cmpr_AST = (AST)currentAST.root;
break;
}
case LT:
{
{
b = (AST)_t;
AST b_AST_in = null;
b_AST = astFactory.create(b);
astFactory.addASTChild(currentAST, b_AST);
match(_t,LT);
_t = _t.getNextSibling();
s += b;
}
cmpr_AST = (AST)currentAST.root;
break;
}
case GE:
{
{
c = (AST)_t;
AST c_AST_in = null;
c_AST = astFactory.create(c);
astFactory.addASTChild(currentAST, c_AST);
match(_t,GE);
_t = _t.getNextSibling();
s += c;
}
cmpr_AST = (AST)currentAST.root;
break;
```

```
}
case LE:
{
{
d = (AST)_t;
AST d_AST_in = null;
d_AST = astFactory.create(d);
astFactory.addASTChild(currentAST, d_AST);
match(_t,LE);
_t = _t.getNextSibling();
s += d;
}
cmpr_AST = (AST)currentAST.root;
break;
}
case EQ:
{
{
e = (AST)_t;
AST e_AST_in = null;
e_AST = astFactory.create(e);
astFactory.addASTChild(currentAST, e_AST);
match(_t,EQ);
_t = _t.getNextSibling();
s += e;
}
cmpr_AST = (AST)currentAST.root;
break;
}
default:
{
throw new NoViableAltException(_t);
}
}
}
catch (RecognitionException ex) {
reportError(ex);
if (_t!=null) {_t = _t.getNextSibling();}
}
returnAST = cmpr_AST;
_retTree = _t;
return s;
}


public static final String[] _tokenNames = {
```

“<0>”,
“EOF”,
“<2>”,
“NULL_TREE_LOOKAHEAD”,
“COLON”,
“SLASH”,
“DOT”,
“PLUS”,
“STAR”,
“ASSIGN”,
“COMMA”,
“LPAREN”,
“RPAREN”,
“LBRKT”,
“RBRKT”,
“EQ”,
“LE”,
“GE”,
“GT”,
“LT”,
“NEQ”,
“REQ”,
“HASSIGN”,
“SEMI”,
“LETTER”,
“DIGIT”,
“ID”,
“INT”,
“STRING”,
“CMNT”,
“WS”,
“\”endif\””,
“\”if\””,
“\”then\””,
“\”for\””,
“\”HOSTS\””,
“\”in\””,
“\”lan_def\””,
“\”host_def\””,
“\”var\””,
“\”@=\””,
“\”%=\””,
“\”<>\””,
“\”rule\””,
“\”TCP\””,
“\”UDP\””,

```
"\"ICMP\"",
"\"report\"",
"\"target=\"",
"\"protocol=\"",
"\"direction=\"",
"\"topology\"",
"\"allow\"",
"\"block\"",
"\"out\"",
"\"snd\"",
"\"rcv\"",
"\">\""
};

private static final long[] mk_tokenSet_0() {
long[] data = { 2524362733273600L, 0L};
return data;
}
public static final BitSet _tokenSet_0 = new BitSet(mk_tokenSet_0());
}
//-----------------------------------------
// Main File for Firedrl  -- Marvin Rich
//-----------------------------------------
import antlr.CommonAST;
//-------------------------------------------------
// Firedrl Language Processor
//-------------------------------------------------
import java.io.*;

class Firedrl {
  public static void main(String[] args) {

    try {

      //-- Check for proper number arguments
      if (args.length != 1) {
        System.err.println("Usage: <inputfile> ");
        System.exit(1);
      }

      //-- Define input file streams
      DataInputStream in =
        new DataInputStream(new FileInputStream(args[0]));

      //-- Invoke Lexer,Parser & Walker
      FiredrlLexer  l = new FiredrlLexer(in);
```

```
      FiredrlParser p = new FiredrlParser(l);
      p.file();
      FiredrlWalker w = new FiredrlWalker();
      w.file((CommonAST) p.getAST());
    }

    catch (Exception e) {
      System.err.println(e);
    }

  }
}

//-- Marvin Rich Firedrl.g file
//---------------------------------------------------
// Firedrl Language Lexer
//---------------------------------------------------
class FiredrlLexer extends Lexer;
options {
  k = 2;
  exportVocab = Firedrl;
}

COLON  : ':' ;
SLASH  : '/' ;
DOT    : '.' ;
PLUS   : '+' ;
STAR   : '*' ;
ASSIGN : '=' ;
COMMA  : ',' ;
LPAREN : '(' ;
RPAREN : ')' ;
LBRKT  : '{' ;
RBRKT  : '}' ;
EQ     : "==" ;
LE     : "<=";
GE     : ">=";
GT     : '>';
LT     : '<';
NEQ    : "@=";
REQ    : "%=";
HASSIGN : "<>";
SEMI   : ';' ;


protected LETTER : ('a'..'z') | ('A'..'Z') ;
```

```
protected DIGIT  : '0'..'9';


ID options { testLiterals = true; }
      : LETTER (LETTER | DIGIT | '_')* ;

INT : (DIGIT)+;
STRING : ""! ("" ""! | ~(""))* ""! ;

CMNT   : "//" (~'\n')* '\n' {newline(); $setType(Token.SKIP);};

WS     : (' ' | '\t' | '\n' { newline(); } | '\r' )
        { $setType(Token.SKIP); } ;

//---------------------------------------------------
// Firedrl Language Parser
//---------------------------------------------------
class FiredrlParser extends Parser;

options {
  buildAST = true;
  k = 2;
}

file  : src_code;



src_code  :
        (EOF! |  stmt SEMI! src_code | cntl_stmt  | RBRKT^  src_code | "endif"!
SEMI! src_code) ;


cntl_stmt :
          "if"^ pred "then"! src_code

        | "for"^  LPAREN! "HOSTS"! "in"! ID  RPAREN! LBRKT!
           src_code
        ;



//-- Firdrl Statement Types
stmt  : "lan_def"^  lan (COMMA! lan)*

        | "host_def"^ ID (COMMA! ID)*
```

```
        | "var"^ ID (COMMA! ID)*

        | ID "@="^ host_intfc (COMMA! host_intfc)*

        | ID "%="^ ID (COMMA! ID)*

        | "<>"^  ID (COMMA! ID)*

        | ID ASSIGN^ var_src

        | "rule"! ID ASSIGN! action direction ip_sel ip_sel
         (   ("TCP"^    (port) (port) INT)
           | ("UDP"^    (port) (port) )
           | ("ICMP"^   INT )
         )

        | "report" "target="! rpt_targ "protocol="! rpt_proto
                "direction="! rpt_dir

        | "report"  "topology"^

       ;

lan      : ID COLON^ ipaddr SLASH! INT;

ipaddr   : INT DOT^ INT DOT! INT DOT! INT ;

port     : ((cmpr)? INT | STAR);

cmpr     : LT | GT | EQ | LE | GE ;

action   : ("allow" | "block");

direction : ("in" | "out");

ip_sel   : (ipaddr | STAR);

rpt_targ  : (STAR | ID);

rpt_proto : (STAR | "TCP" | "UDP" | "ICMP");

rpt_dir   : (STAR | "snd" | "rcv" );

host_intfc  : ID COLON! INT;
```

```
var_src   : (ID | INT);

pred      : ID ">"^ INT;

//--------------------------------------------------
// Firedrl Language Walker
//--------------------------------------------------
{
  import java.io.* ;
  import java.util.* ;
  import antlr.CommonAST;

}
class FiredrlWalker extends TreeParser;

options {
  buildAST = true;
}

{
  java.util.Hashtable dict = new java.util.Hashtable();


  //-----------------------------------------
  // Lan class contains the definition of a
  // single lan segment.
  //-----------------------------------------
  public class Lan {
    String name;       // Lan Name
    String ipaddr;     // Lan IP addr in dot notation
    int    netmsk;     // Mask for Lan

    //-- Contains hosts on this lan
    Hashtable hosts = new Hashtable();
  }

  //-----------------------------------------
  // Host class contains the name, interfaces,
  // and Firewall rules for a host.
  //-----------------------------------------
  public class Host {

    String name;               //host name

    //-- Contains Lan Interfaces assocaited with this host
    java.util.Hashtable intfc = new java.util.Hashtable();
```

```
  //-- Contains Firewall rules for this host
  java.util.Hashtable rules = new java.util.Hashtable();

}


//-----------------------------------------
// Intfc class defines each Lan interface to
// a host (e.g. similar to NIC connection)
//-----------------------------------------
public class Intfc {
  String lname;     //Lan name
  String hname;     //Host name
  int ID;           //Host ID on Lan
}

//-----------------------------------------
// Var class for each variable used.
//-----------------------------------------
public class Var {
  String name;      //variable  name
  int val;          //variable  value
}

//-----------------------------------------
// Rule class contains the definition of a
// single firewall rule.
//-----------------------------------------
public class Rule {
  String name;       // Rule Name
  String type;       // Rule Type (TCP,UDP or ICMP)
  String action;     // access action
  String direction;  // access direction
  String src_addr;   // src address
  String dest_addr;  // dest address
  String src_port;   // src port
  String dest_port;  // dest port
  int    icmp_type;  // icmp type
  int    tcp_con;    // tcp connect flags
}

//-----------------------------------------
// TestNetWork class gets populated via
// network definition statements of language
//-----------------------------------------
```

```
public class TestNetWork  {

  Hashtable  L = new Hashtable();
  Hashtable  H = new Hashtable();
  Hashtable  R = new Hashtable();
  Hashtable  V = new Hashtable();

  //-- Name existance check
  void isExist(String s) {

    if (runnable) {
      if (  !L.containsKey(s) &&
            !H.containsKey(s) &&
            !R.containsKey(s) &&
            !V.containsKey(s) ) {
        runnable = false;
        System.out.println("Error: Name:"+ s + " undefined!");
      }
    }

  }

  //-- Name existance check
  void noExist(String s) {

    if (runnable) {
      if (  L.containsKey(s) | |
            H.containsKey(s) | |
            R.containsKey(s) | |
            V.containsKey(s) ) {
        runnable = false;
        System.out.println("Error: Name:"+ s + " already defined!");
      }
    }

  }

  //-- Add Lan segment definition
  void addLan(Lan l) {

    //-- Check name not already defined
    noExist(l.name);

    //-- Process if no errors
    if (runnable)
      L.put(l.name,l);
```

```
}

//-- Add Host definition
void addHost(String hname) {

  //-- Check name not already defined
  noExist(hname);

  //-- Process if no errors
  if (runnable) {
   Host h = new Host();
   h.name = hname;
   H.put(hname,h);
  }
}

//-- Add Host to Lan Definition
void addHostToLan(Intfc i) {

  Lan  l = new Lan();
  Host h = new Host();

  //-- Check names exist and of proper type
  isExist(i.hname);
  isExist(i.lname);
  if (runnable) {
    if (!H.containsKey(i.hname)) {
      runnable = false;
      System.out.println("Error: Name: "+ i.hname + " not host name type!");
    }
    if (!L.containsKey(i.lname)) {
      runnable = false;
      System.out.println("Error: Name: "+ i.lname + " not lan name type!");
    }
  }

  //-- Process if no errors
  if (runnable) {

    //-- Link host to Lan
    l = (Lan) L.get(i.lname);
    l.hosts.put(i.hname,i.hname);
    L.put(l.name,l);

    //-- Add interface to host
    h = (Host) H.get(i.hname);
```

```
    h.intfc.put(i.lname,i);
    H.put(h.name,h);

  }

}



//-- Add Rule to Host Definition
void addRuleToHost(String hname, String rname) {


  //-- Check names exist and of proper types
  isExist(hname);
  isExist(rname);
  if (runnable) {
    if (!H.containsKey(hname)) {
      runnable = false;
      System.out.println("Error: Name "+ hname + " not host name type!");
    }
    if (!R.containsKey(rname)) {
      runnable = false;
      System.out.println("Error: Name "+ rname + " not rule name type!");
    }
  }

  //-- Process if no errors
  if (runnable) {
    Host h = new Host();
    h = (Host) H.get(hname);
    h.rules.put(rname,rname);
    H.put(h.name,h);
  }

}


//-- Add variable
void addVar(String vname) {

  //-- Check variable not defined already
  noExist(vname);

  //-- Process if no errors
  if (runnable) {
```

```
    Var tmp = new Var();
    tmp.name = vname;
    tmp.val  = 0;
    V.put(vname, tmp);
  }

}

//-- Assign one variable to another
void assignVar(String lname,String rname) {

  //-- Check names exist and of proper type
  isExist(lname);
  isExist(rname);
  if (runnable) {
    if (!V.containsKey(lname)) {
      runnable = false;
      System.out.println("Error: Name "+ lname + " not var type!");
    }
    if (!V.containsKey(rname)) {
      runnable = false;
      System.out.println("Error: Name "+ rname + " not var type!");
    }
  }

  //-- Process if no errors
  if (runnable) {
    Var v1 = new Var();
    Var v2 = new Var();
    v1 = (Var) V.get(rname);
    v2 = (Var) V.get(lname);
    v1.val = v2.val;
    V.put(rname,v1);
  }

}

//-- Assign constant to a variable
void assignVarConst(String vname, int val) {

  isExist(vname);
  if (runnable) {
    if (!V.containsKey(vname)) {
      runnable = false;
      System.out.println("Error: Name "+ vname + " not var type!");
    }
```

```
  }

  //-- Process if no errors
  if (runnable) {
    Var v = new Var();
    v = (Var) V.get(vname);
    v.val = val;
    V.put(vname,v);
  }

}

//-- Supports loop through Lan hosts
boolean stillHosts(Enumeration e1) {

  if (e1.hasMoreElements()) {
    cur_host = (String) e1.nextElement();
    return true;
  }
  else
    return false;

}


//-- Add Rule Definition
void addRule(String rname,String b,String c, String d,
             String e, String f, String g, String h,
             int i, int j) {

  //-- Check rule name not defined already
  noExist(rname);

  //-- Process if no errors
  if (runnable) {
    Rule r  = new Rule();
    r.name= rname; r.type = b; r.action = c; r.direction = d;
    r.src_addr = e; r.dest_addr = f; r.src_port = g; r.dest_port = h;
    r.icmp_type = i; r.tcp_con = j;
    R.put(r.name,r);
  }
}

//-- Report Network Topology
void reportNetwork() {
```

```
System.out.println("+------------------------------");
System.out.println("+ Lan Definitions ");
System.out.println("+------------------------------");

for (Enumeration e1 = L.elements();
    e1.hasMoreElements();  ) {
  Lan l = (Lan) e1.nextElement();
  System.out.println(" Lan Name = "  + l.name +
                  " IP Addr =  "  + l.ipaddr +
                  " Net Msk =  "  + l.netmsk);

  for (Enumeration e2 = l.hosts.elements();
      e2.hasMoreElements();  ) {
    String s = (String) e2.nextElement();
    System.out.println("  -- Host= " + s);
  }
}

System.out.println("+------------------------------");
System.out.println("+ Host Definitions ");
System.out.println("+------------------------------");

for (Enumeration e1 = H.elements();
    e1.hasMoreElements();  ) {
  Host h = (Host) e1.nextElement();
  System.out.println(" Host Name = "  + h.name );

  for (Enumeration e2 = h.intfc.elements();
      e2.hasMoreElements();  ) {
    Intfc i = (Intfc) e2.nextElement();
    System.out.println(" --- Lan Name = " + i.lname);
    System.out.println("     Lan ID = " + i.ID);
  }

  for (Enumeration e2 = h.rules.elements();
      e2.hasMoreElements();  ) {
    String s = (String) e2.nextElement();
    System.out.println(" --- Rule Name = " + s);
  }
}

System.out.println("+------------------------------");
System.out.println("+ Rule Definitions ");
System.out.println("+------------------------------");
for (Enumeration e = R.elements();
    e.hasMoreElements();  ) {
```

```
        Rule r = (Rule) e.nextElement();

        System.out.println(" Rule name = "  + r.name);
        System.out.println(" ---- type = "  + r.type +
                           " act =  "       + r.action +
                           " dir = "        + r.direction);
        System.out.println("      src =  "       + r.src_addr +
                           " s_port = "     + r.src_port +
                           " dest = "       + r.dest_addr +
                           " d_port = "     + r.dest_port +
                           " i_type = "     + r.icmp_type +
                           " con ="         + r.tcp_con );
      }
    }
  }

  boolean runnable = true;              //False if semantic errors found
  String cur_host = "C";                //Current host name within loop
  TestNetWork ntwk = new TestNetWork();  //Create Test Network Environment

}

//-----------------------------------------------
// Start of Firedrl AST walker, which generates
// the java code to process Firedrl language
//-----------------------------------------------

file  {int r;} : (r=stmt)+ ;



stmt returns [int r=0;] {
      Lan    l = new Lan();
      Host   h = new Host();
      Intfc  i = new Intfc();
      String s1,s2,p1,p2;
      String act,dir,name;
      Intfc  i2;
     }

     : #("lan_def"   (l=lan {ntwk.addLan(l); r = 0;} )+ )

     | #("host_def"  (ID { ntwk.addHost(#ID.getText()); r = 0;} )+ )

     | #("var"       (ID    {ntwk.addVar(#ID.getText()); r = 0;} )+ )
```

```
| #("@=" ID {i.lname = #ID.getText();}
   (i2=intfc[i] {ntwk.addHostToLan(i2); r = 0;})+ )

| #("%=" d:ID (ID {ntwk.addRuleToHost(d.getText(),#ID.getText()); r=0;})+ )

| #("<>" (ID {ntwk.addRuleToHost(cur_host,#ID.getText()); r=0;})+ )

| #("if" s1=p:pred   {
     AST thenpart = p.getNextSibling();
     System.out.println("At if statement thenpart");
     if (true) r=stmt(thenpart);
     r=0;
   }
  )

|  #("for" a:ID
     {
      l = (Lan) ntwk.L.get(a.getText());
      Enumeration e1 = l.hosts.elements();
      while (ntwk.stillHosts(e1)) {
        AST strt_of_block = a.getNextSibling();
        stmt(strt_of_block);
      }
     }
   )

| #(RBRKT  {
        l = (Lan) ntwk.L.get(a.getText());
        Enumeration e1 = l.hosts.elements();
        while (ntwk.stillHosts(e1)) {
          AST strt_of_block = a.getNextSibling();
          stmt(strt_of_block);
        }
       }
   )


| #(ASSIGN  ID var_src[#ID.getText()] {r = 0;})


| #("TCP" n1:ID act=action dir=direction
   s1=ip_sel s2=ip_sel p1=port p2=port c1:INT
   {ntwk.addRule(n1.getText(),"TCP",act,dir,s1,s2,p1,p2,
            0,Integer.parseInt(c1.getText())); r=0;} )

| #("UDP" n2:ID act=action dir=direction
```

```
        s1=ip_sel s2=ip_sel p1=port p2=port
        {ntwk.addRule(n2.getText(),"UDP",act,dir,s1,s2,p1,p2,0,0); r=0;} )

    | #("ICMP" n3:ID act=action dir=direction
        s1=ip_sel s2=ip_sel c2:INT
        {ntwk.addRule(n3.getText(),"ICMP",act,dir,s1,s2,"","",
                Integer.parseInt(c2.getText()),0); r=0;} )

    | #("report" s1=rpt_targ p1=rpt_proto s2=rpt_dir {r=0;})

    | #("topology"  {ntwk.reportNetwork(); r=0;})

    ;

pred returns [String s="";] :#(">"  ID  {s = #ID.getText();}
                        {s += ">";}
                        b:INT  {s += b;}
                    )
                    ;

lan returns [Lan lan = new Lan();]
   {String s="";}
   :  #(COLON a:ID {lan.name = a.getText();}
               s=ipaddr {lan.ipaddr = s;}
               b:INT {lan.netmsk = Integer.parseInt(b.getText());}  )
   ;


ipaddr returns [String s = "";]
   : #(DOT a:INT b:INT c:INT d:INT {s = a+"."+b+"."+c+"."+d;} )
   ;

port returns [String s = "";]
    {String a;}
   : ((a=cmpr {s += a;})? b:INT {s += b;} | c:STAR {s += c;})
   ;

action returns [String s = "";]
   : ( (a:"allow" {s += a;}) | (b:"block" {s += b;}) )
   ;

ip_sel returns [String s = "";]
   {String a;}
   : ( ( a=ipaddr {s += a;}) | (b:STAR {s += b;}) )
   ;
```

```
direction returns [String s = “”;]
   : ( (a:”in” {s += a;}) | (b:”out” {s += b;}) )
   ;

cmpr returns [String s = “”;]
   :   (a:GT {s += a;}) | (b:LT {s += b;}) |
       (c:GE {s += c;}) | (d:LE {s += d;}) |
       (e:EQ {s += e;})
   ;

var_src [String s]
   : (   (ID {ntwk.assignVar(s,#ID.getText());})
       | (b:INT {ntwk.assignVarConst(s,Integer.parseInt(b.getText()));})
     )
   ;

rpt_targ returns [String s = “”;]
   : ( ( a:ID {s += a.getText();}) | (b:STAR {s += b;}) )
   ;

rpt_proto returns [String s = “”;]
   : ( ( a:”TCP” {s += a;}) | ( b:”UDP” {s += b;}) | ( c:”ICMP” {s += c;})
       | (d:STAR {s += d ;})
     )
   ;

rpt_dir returns [String s = “”;]
   : ( ( a:”snd” {s += a;}) | ( b:”rcv” {s += b;}) | ( c:STAR {s += c;}) )
   ;

intfc [Intfc i2] returns [Intfc i = new Intfc();]
   {i.lname = i2.lname;}
   : ID     {i.hname = #ID.getText();}
     a:INT  {i.ID = Integer.parseInt(a.getText());}
   ;
```