NPS-CS-02-001

NAVAL POSTGRADUATE SCHOOL
Monterey, California

KeyNote Policy Files and Conversion to Disjunctive Normal Form for Use in IPsec

by

Evdoxia Spyropoulou, Timothy E. Levin, Cynthia E. Irvine

January 2002

Approved for public release; distribution is unlimited.

Prepared for: Defense Advanced Research Project Agency
Advanced Technology Organization

NAVAL POSTGRADUATE SCHOOL
Monterey, California 93943-5000

RADM Admiral David R. Ellison Superintendent	R. Elster Provost

This report was prepared for and funded by the Naval Postgraduate School Center for Information Systems Security (INFOSEC) Studies and Research (NPS CISR) .

This report was prepared by:

Evdoxia Spyropoulou
Evdoxia Spyropoulou
Research Associate

Timothy E Levin
Timothy E. Levin
Research Associate Professor

Cynthia E. Irvine
Cynthia E. Irvine
Associate Professor

Reviewed by:

Neil C. Rowe
Neil C. Rowe
Professor
Department of Computer Science

Christopher Eagle, Chair
Department of Computer Science

Released by:

D. W. Netzer
Associate Provost and
Dean of Research

REPORT DOCUMENTATION PAGE	Form approved OMB No 0704-0188

**Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources,
gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.**

1. AGENCY USE ONLY (Leave blank)	**2. REPORT DATE** January 2002	**3. REPORT TYPE AND DATES COVERED** Technical Report

4. TITLE AND SUBTITLE KeyNote Policy Files and Conversion to Disjunctive Normal form for Use in IPsec	**5. FUNDING** MIPR No. 00-E583
6. AUTHOR(S) Evdoxia Spyropoulou, Timothy E. Levin and Cynthia E. Irvine	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Center for Information Systems Security Studies and Research (CISR) Naval Postgraduate School, 833 Dyer Road, Monterey, CA 93943	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) DARPA/ATO CHATS Program 3701 North Fairfax Drive Arlington, VA 22203-1714	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT (Maximum 200 words.)

We describe a utility for converting a KeyNote policy file to Disjunctive Normal Form, so that it can be further utilized in our research on Quality of Security Service for IPsec. We also provide background information on KeyNote and IPsec, on the Disjunctive Normal Form of logical expressions, as well as on the `lex` and `yacc` tools employ by our utility.

14. SUBJECT TERMS KeyNote, IPsec, computer security, communications security, quality of service, security policy	**15. NUMBER OF PAGES** 46
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** Unclassified

KeyNote Policy Files and Conversion to Disjunctive Normal Form for Use in IPsec

Evdoxia Spyropoulou **Timothy Levin** **Cynthia Irvine**

Naval Postgraduate School
Monterey, CA

1. Introduction

In this technical report we describe the utility we developed for converting a KeyNote policy file to Disjunctive Normal Form, so that it can be further utilized in our research on Quality of Security Service for IPsec. Some background information on KeyNote and IPsec, on the Disjunctive Normal Form of logical expressions, as well as on `lex` and `yacc` tools, which we employ in our utility, can be found in paragraphs below.

2. KeyNote and its use in IPsec

2.1 KeyNote background

Details on the syntax of KeyNote policy files can be found in [1], [2], [3]. The paragraphs below are taken from [2], [3].

KeyNote is a simple and flexible trust-management system designed to work well for a variety of large- and small- scale Internet-based applications. It provides a single, unified language for both local policies and credentials. KeyNote policies and credentials, called *assertions*, contain predicates that describe the trusted actions permitted by the holders of specific public keys. KeyNote assertions are essentially small, highly-structured programs. A signed assertion, which can be sent over an untrusted network, is also called a *credential assertion.* Credential assertions, which also serve the role of certificates, have the same syntax as policy assertions but are also signed by the principal delegating the trust.

In KeyNote, the authority to perform trusted actions is associated with one or more principals. A *principal* may be a physical entity, a process in an operating system, a public key, or any other convenient abstraction. Principals perform two functions of concern to KeyNote: They request `actions' and they issue `assertions.' *Actions* are any trusted operations that an application places under KeyNote control. Actions are described to the KeyNote compliance checker in terms of a collection of name-value pairs called an *action attribute set.* The action attribute set is created by the invoking application. Assertions delegate the authorization to perform actions to other principals.

KeyNote provides advice to applications on the interpretation of policy with regard to specific requested actions. Applications invoke the KeyNote compliance checker by issuing a *query* containing a proposed action attribute set and identifying the principal(s) requesting it. The KeyNote system determines and returns an appropriate *policy compliance value* from an ordered set of possible responses. The policy compliance value returned from a KeyNote query advises the application how to process the requested action. In the simplest case, the compliance value is Boolean (e.g., "reject" or "approve").

Assertions are the basic programming unit for specifying policy and delegating authority. Assertions describe the conditions under which a principal authorizes actions requested by other principals. An assertion identifies the principal that made it, which other principals are being authorized, and the conditions under which the authorization applies.

A special principal, whose identifier is "POLICY", provides the root of trust in KeyNote. "POLICY" is therefore considered to be authorized to perform any action. Assertions issued by the "POLICY" principal are called *policy assertions* and are used to delegate authority to otherwise untrusted principals. The KeyNote security policy of an application consists of a collection of policy assertions.

KeyNote assertions are divided into sections, called *fields* that serve various semantic functions. One mandatory field is required in all assertions: Authorizer. Six optional fields may also appear: Comment, Conditions, KeyNote-Version, Licensees, Local-Constants, and Signature.

- The *Authorizer* identifies the Principal issuing the assertion.
- The *Comment* field allows assertions to be annotated with information describing their purpose.
- The *Conditions* field gives the `conditions' under which the Authorizer trusts the Licensees to perform an action. `Conditions' are predicates that operate on the action attribute set.
- The *KeyNote-Version* field identifies the version of the KeyNote assertion language under which the assertion was written.
- The *Licensees* field identifies the principals authorized by the assertion. More than one principal can be authorized, and authorization can be distributed across several principals through the use of `and' and threshold constructs.
- The *Local-Constants* field adds or overrides action attributes in the current assertion only.
- The *Signature* field identifies a signed assertion and gives the encoded digital signature of the principal identified in the Authorizer field.

Action attributes provide the primary mechanism for applications to pass information to assertions. Attribute names are strings from a limited character set, and attribute values are represented internally as strings.

2.2 KeyNote in IPsec

IPsec provides security services including confidentiality, integrity, authenticity, through the establishment of Security Associations (SA) among the entities that wish to communicate. The SA is a "simplex connection that affords security services to the traffic carried by it" and it essentially is "a management construct used to enforce a security policy in the IPsec environment" [4]. There is a set of parameters associated with each SA, which includes, among others: SA lifetime, encryption and/or authentication algorithms and keys, and protocol mode (tunnel/transport). The SAs can be generated manually, but that approach does not scale well. The Internet Key Exchange (IKE) along with the Internet Security Association and Key Management Protocol (ISAKMP) address the problem of establishing and maintaining SAs through the use of an automated daemon.

The IPsec protocols themselves do not include an approach for managing the policies that control which host is allowed to establish SAs with another host and what kind of characteristics the SAs should have. We are using the OpenBSD's implementation of IPsec [5]. This implementation addresses the SA management problem by including the KeyNote trust management system and providing an additional check in the IPsec processing: it makes sure that the SAs to be created agree with a local security policy (that can be expressed in KeyNote's language).

KeyNote is used in OpenBSD for enforcing the local policy that controls which host is allowed to establish SAs with another host and what kind of characteristics the SAs should have. When two IKE daemons negotiate for establishing an SA, the initiator sends across proposals for the SAs he is willing to establish. As mentioned in [6] "IKE proposals are "suggestions" by the initiator of an exchange to the responder as to what protocols and attributes should be used on a class of packets. For example, a given exchange may ask for ESP with 3DES and MD5 and AH with SHA1 (applied successively on the same packet), or just ESP with Blowfish and RIPEMD-160. The responder examines the proposals and determines which of them are acceptable, according to policy and any credentials. The goal of security policy for IKE is thus to determine, based on local policy, credentials provided during the IKE exchanges (or obtained through other means), the SA attributes proposed during the exchange, and perhaps other (side-channel) information, whether a pair of SAs should be installed in the system (in fact, whether both the IPsec SAs should be installed). For each proposal suggested by or to the remote IKE daemon, the KeyNote system is consulted as to whether the proposal is acceptable based on local policy and remote credentials (e.g., KeyNote credentials or X509 certificates provided by the remote IKE daemon)". The local policy is contained in the `isakmpd.policy` file, which is simply a flat ASCII file containing KeyNote policy assertions.

The responder selects a proposal, the first one from the list of proposals that are sent to him that conforms to his local policy (as expressed in `isakmpd.policy`). He sends this proposal back to the initiator. The initiator checks his own `isakmpd.policy`, to make sure that the selected proposal indeed agrees with his local policy.

Briefly, KeyNote policy assertions used in IKE have the following characteristics as described in [6]:

- The Authorizer field is typically "POLICY".
- The Licensees field can be an expression of pass phrases used for authentication of the Main Mode exchanges, and/or public keys (typically, X509 certificates), and/or X509 Canonical names.
- The Conditions field contains an expression of attributes from the IPsec policy action set (see below for more details).
- The ordered return-values set for IPsec policy is "false, true".

Information about the proposals, the identity of the remote IKE daemon, the packet classes to be protected, etc. are encoded in what is called an *action set.* The action set is composed of name-value attributes, similar in some way to a shell environment variable. These values are initialized by the IKE daemon before each query to the KeyNote system, and can be tested against in the Conditions field of assertions. Note that assertions and credentials can make reference to non-existent attributes without catastrophic failures (access may be denied, depending on the overall structure, but will not be accidentally granted). One reason for credentials referencing non-existent attributes is that they were defined within a specific implementation or network only.

The action attributes that are currently defined for IPsec can be seen in Figure 1 and more details on them and their values can be found in [6].

```
app_domain
doi
initiator
phase_1
pfs
ah_present, esp_present, comp_present
ah_hash_alg
esp_enc_alg
comp_alg
```

```
ah_auth_alg
esp_auth_alg
ah_life_seconds, esp_life_seconds, comp_life_seconds
ah_life_kbytes, esp_life_kbytes, comp_life_kbytes
ah_encapsulation, esp_encapsulation, comp_encapsulation
comp_dict_size
comp_private_alg
ah_key_length, esp_key_length
ah_key_rounds, esp_key length
ah_group_desc, esp_group_desc, comp_group_desc
phase1_group_desc
remote_filter_type, local_filter_type, remote_id_type
remote_filter_addr_upper, local_filter_addr_upper, remote_id_addr_upper
remote_filter_addr_lower, local_filter_addr_lower, remote_id_addr_lower
remote_filter, local_filter, remote_id
remote_filter_port, local_filter_port, remote_id_port
remote_filter_proto, local_filter_proto, remote_id_proto
remote_negotiation_address
local_negotiation_address
GMTTimeOfDay
LocalTimeOfDay
```

Figure 1: IPsec Action Attributes

In Figure 2 an example of a typical `isakmpd.policy` file can be found.

```
Keynote-version: 2
Licensees: "passphrase:mekmitasdigoat" || "x509-base64:abcd=="
Comment: This policy accepts anyone using shared-secret
         authentication with the password mekmitasisgoat, or the
         public key contained in the X509 certificate encoded as
         "abcd==", as long as he does ESP only (no AH) using perfect
         forward secrecy with either 3DES or IDEA.
Authorizer: "POLICY"
Conditions: app_domain == "IPsec policy" && doi == "ipsec" &&
       pfs == "yes" && esp_present == "yes" && ah_present == "no" &&
       (esp_enc_alg == "3des" || esp_enc_alg == "idea") -> "true";
```

Figure 2: A typical `isakmpd.policy` file

3. Disjunctive Normal Form of KeyNote policy files

3.1 Definition of Disjunctive Normal Form

A boolean expression is an expression involving variables each of which can take on either the value true or the value false. These variables are combined using boolean operations such as AND (conjunction), OR (disjunction), and NOT (negation) [7].

A statement is in Disjunctive Normal Form (DNF) if it is a disjunction (sequence of ORs) consisting of one or more disjuncts, each of which is a conjunction (AND) of one or more literals (i.e., statement letters and negations of statement letters) [8].

Examples of disjunctive normal forms include:

(A && B) || (!A && C)

(A && C) || (B && C)

(A && B && !C) || (!B && C) || (B && C && !D)

where the symbols !, &&, ||, denote the logical NOT, AND, OR respectively.

Every expression in logic consisting of a combination of multiple &&, ||, and !s can be written in disjunctive normal form.

3.2 General algorithm for conversion to DNF

By systematically applying the laws of Boolean algebra, a disjunctive normal form for any Boolean expression can be computed as follows [9]:

(i) If any negation appears outside any parenthesis in the expression, move it inside by applying de Morgan laws

!(A && B) = !A || !B

!(A || B) = !A && !B

This way the expression now involves letters and negations of letters combined by ANDs and ORs.

(ii) Use the distributive identities to create a disjunction of conjunctions of literals

A || (B && C) = (A || B) && (A || C)

A && (B || C) = (A && B) || (A && C)

(iii) If any conjunction contains both a letter and its negation, it can be dropped.

3.3 Motivation for conversion of policy files to DNF

The KeyNote policy file `isakmpd.policy` contains the acceptable values for the IPsec parameters. Our policy may accept more than one value for some of the attributes. For example, the policy file:

```
KeyNote-Version: 2
Authorizer: "POLICY"
Licensees: "passphrase:mekmitasdigoat"
Conditions: app_domain == "IPsec policy" &&
          ( (esp_present == "yes") &&
            ( (esp_enc_alg == "des") || (esp_enc_alg == "3des") ) &&
            ( (esp_auth_alg == "hmac-md5") || (esp_auth_alg == "hmac-sha") )
          ) -> "true";
```

accepts IKE proposals for encryption with the ESP protocol as long as the encryption algorithm is one of: DES, 3DES and the authentication algorithm one of: MD5, SHA. This policy file accepts any value for the rest of the IPsec SA attributes.

The DNF form of the Conditions field is:

```
( (app_domain == "IPsec policy") && (esp_present == "yes") &&
  (esp_enc_alg == "des") && (esp_auth_alg == "hmac-md5") )
||
( (app_domain == "IPsec policy") && (esp_present == "yes") &&
  (esp_enc_alg == "des") && (esp_auth_alg == "hmac-sha") )
||
( (app_domain == "IPsec policy") && (esp_present == "yes") &&
  (esp_enc_alg == "3des") && (esp_auth_alg == "hmac-md5") )
```

```
||
( (app_domain == "IPsec policy") && (esp_present == "yes") &&
  (esp_enc_alg == "3des") && (esp_auth_alg == "hmac-sha") )
```

So in the DNF form each conjunction describe a possible combination of IPsec SA attributes and values that a valid SA proposal can contain.

The DNF form is useful for constructing SA proposals from it. Currently the IKE daemon retrieves SA proposals from a configuration file, sends them across to the IKE peer, and checks whether the peer's selected proposal agrees to the policy in `isakmpd.policy`. So local policy is represented in two areas: the daemon's configuration file and KeyNote policy file. This causes a problem in the area of security policy management. We have modified the IKE process so that it retrieves information for the proposals from the KeyNote policy file. The policy file in DNF contains the set of all acceptable proposals, although all the SA characteristics may not be described. It is a straightforward process though to construct the full proposals that will be sent to the peer by using default values for the IPsec attributes not mentioned in the policy files.

Furthermore the DNF form of the KeyNote policy file facilitates another area of our work, which is briefly described below.

We have introduced the notion of Quality of Security Service (QoSS) which refers to the ability to provide *security* services according to user and system preferences and policies [10]. The enabling technology for both QoSS and a security-adaptable infrastructure is variant security, or the ability of security mechanisms and services to allow the amount, kind or degree of security to vary, within predefined ranges.

We have described how variant security can be offered and presented to applications and users in an organized manner [11]. Two abstractions were introduced:

- an operational mode parameter, *Network Mode*, which represents the influence external conditions and network status could have on the security policy and security services applicable to a task: for example under certain conditions, an administrator may be willing to accept more (or less) security for a given application. Example values for this parameter are: "normal", "impacted", "emergency".
- a *Security Level* parameter, which represents the choices available to users within the ranges permitted from the policy for the security variables. Example values for this parameter are: "high", "medium", "low".

We are currently working on modulating the IPsec security mechanism to provide different levels for security in response to QoSS requests from users [12], [13]. This way we link QoSS conditions to IPsec, so that we can adjust the kind of security services provided to applications according to QoSS "handles", like the network mode and/or the security level.

We modified the IKE daemon and KeyNote, to include in the KeyNote action set the QoSS attributes `network_mode` and `security_level` [14]. This way the policy file may describe more complex security policies that accept different characteristics for the SAs depending on the current system status. An example of such a KeyNote policy file with the QoSS parameters can be found in paragraph 5.

If the newly introduced QoSS attributes and the respective authorized SA "attribute- attribute value" pairs are represented in the Conditions field of KeyNote, the proposals that we are willing to accept and that should be sent to IKE peers depend on the current values `network_mode` and `security_level`. If we use the modified IKE process mentioned above, we should be able to select from the set of all possible SA proposals the ones that are valid for the current system state. The DNF form of the KeyNote policy file facilitates this selection (details for how this proccssing is donc can bc found in [14]).

4. Using lex and yacc for Conversion to DNF

lex and yacc are tools for imposing structure on the input to a program. lex is a lexical analyzer generator and yacc is a parser generator. lex programs recognize regular expressions and pick up the basic items (*tokens*) from the input stream. yacc generates parsers that accept a large class of context-free grammars and organize the tokens according to the input structure rules [15].

We use these tools to take an input KeyNote policy expression and organize it into the disjunctive normal form.

4.1 lex Background

The following paragraphs are taken from [16].

lex is a tool for generating *scanners*: programs which recognize lexical patterns in text. lex reads the given input file for a description of a scanner to generate. The description is in the form of pairs of regular expressions and C code, called *rules*. When the scanner is run, it analyzes its input for occurrences of the regular expressions. Whenever it finds one, it executes the corresponding C code.

The lex input file consists of three sections, separated by a line with just `%%' in it:

```
definitions
%%
rules
%%
user code
```

The *definitions* section contains declarations of simple *name* definitions to simplify the scanner specification, and declarations of *start conditions*. Name definitions have the form:

```
name definition
```

For example,

```
DIGIT    [0-9]
ID       [a-z][a-z0-9]*
```

defines "DIGIT" to be a regular expression which matches a single digit, and "ID" to be a regular expression which matches a letter followed by zero-or-more letters-or-digits.

The *rules* section of the lex input contains a series of rules of the form:

```
pattern   action
```

Finally, the user code section is used for companion routines, which call or are called by the scanner.

When the generated scanner is run, it analyzes its input looking for strings, which match any of its patterns. If it finds more than one match, it takes the one matching the most text. If it finds two or more matches of the same length, the rule listed first in the lex input file is chosen.

Once the match is determined, the text corresponding to the match -the token- is made available in the global character pointer yytext, and its length in the global integer yyleng. The *action* corresponding to the matched pattern is then executed (a more detailed description of actions follows), and then the remaining input is scanned for another match.

Each pattern in a rule has a corresponding action, which can be any arbitrary C statement. If the action is empty, then when the pattern is matched the input token is simply discarded.

Actions can include arbitrary C code, including `return` statements to return a value to whatever routine called the scanner. Each time the scanner is called it continues processing tokens from where it last left off until it either reaches the end of the file or executes a return. Actions are free to modify `yytext`.

`lex` provides a mechanism for conditionally activating rules. Any rule whose pattern is prefixed with "<sc>" will only be active when the scanner is in the start condition named "sc". For example,

```
<STRING>[^"]*         { /* eat up the string body ... */
            ...
            }
```

will be active only when the scanner is in the "STRING" start condition.

Start conditions are declared in the definitions section of the input. A start condition is activated using the `BEGIN` action. Until the next `BEGIN` action is executed, rules with the given start condition will be active and rules with other start conditions will be inactive.

One of the main uses of `lex` is as a companion to the `yacc` parser-generator. `yacc` parsers expect to call a routine named `yylex()` to find the next input token. The routine is supposed to return the type of the next token as well as putting any associated value in the global `yylval`.

REMARK:
`lex` has the option `` `-Pprefix' `` which changes the default `` `yy' `` prefix used by `lex` for all globally-visible variable and function names to instead be *prefix*. In our case we use `` `-Pkd' `` so for example the name of `yytext` is changed to `` `kdtext' ``.

4.2 Using **`lex`** in KeyNote

The `lex` file of our utility can be found in Appendix A. `lex` was already used in KeyNote code for processing of KeyNote files. For the purposes of our research we modified the file `keynote.l` [17]. What follows is a description of the key points of our lex program and its differences from `keynote.l`:

First of all it should be noted that in this work we only deal with the Conditions field of the KeyNote files. The other fields do not participate in the DNF of the policy expression.

In the *definitions* section of our `lex` file the same name definitions were used as in `keynote.l`. These regular expressions define what string sequences we consider to be digits, numbers, variables, literals etc.

The start conditions though are differentiated. We only need to process lexically the Conditions field of the KeyNote policy file. Therefore we have two start conditions:

- `OUTCOND` which indicates that we are outside the Conditions field and
- `CONDITIONS` which means that we are in the Conditions field and we should be analyzing the lexical patterns when we are in this state.

In the *rules* section of the `lex` file:

A set of rules defines how we alternate between start conditions. When we are in `OUTCOND`, a match of the keyword `Conditions` in the KeyNote policy file, will put us in `CONDITIONS` start condition.

In `CONDITIONS` start condition, occurrence of the keywords `KeyNote-Version`, `Comment`, `Authorizer`, `Licensees`, `Signature` indicates that a new KeyNote policy file field is beginning, so we're done with Conditions field and we go to `OUTCOND`.

A set of rules active only in the `CONDITIONS` start condition analyze the strings of KeyNote policy file to match them with the defined lexical patterns and returns a specific token to the `yacc` program.

The strings that we are looking for and the token name returned can be seen below in Table 1:

&&	AND
||	OR
!	NOT
==	EQ
!=	NE
<	LT
>	GT
<=	LE
>=	GE
~=	REGEXP
true	TRUE
false	FALSE
.	DOTT
$	DEREF
(	OPENPAREN
)	CLOSEPAREN
{	OPENBLOCK
}	CLOSEBLOCK
->	HINT
;	SEMICOLON

Table 1: Strings and Token Names in `keynote-dnf.l`

When a variable or a literal string is met (regular expressions in the definitions section define what is considered a variable and a literal in KeyNote policy files), its value is put in `kdlval.string` and the name `VARIABLE` or `STRING` respectively is returned.

Finally rules active independently of the start condition specify that comments, tabs and new lines should be ignored during the lexical processing of the KeyNote policy file.

The user code section contains companion routines, which call or are called by the scanner.

The function `convert_to_dnf()` activates `lex` and `yacc` parsing through the call to `kdparse()` and it is the one that should be called for converting a KeyNote policy file to DNF. It accepts as inputs the file to be converted to DNF and the file where the DNF form will be stored.

NOTE 1: Some of the patterns that can be met in a KeyNote policy file and are processed in `keynote.l`, are not typically met in IPsec policy files and they were ignored in the rules section for the purposes of this work. For completeness though they should be addressed in future versions. They mainly deal with numerical expressions and operations and they are:
`-, *, /, %, ^, @, &, flt, number`

NOTE 2: In this version of our work there is incomplete handling of error flags, error numbers and respective return values. These issues should be addressed in the future.

NOTE 3: The functions `mystrncpy()`, `get_octal()`, `is_octal()` were copied from `keynote.l`. They also exist identical in `keynote-ver.l` [18], so it seemed best to also copy them in our lex file. Maybe there's a better way for doing this, instead of keeping separate functions in every `lex` file.

4.3 yacc Background

The following paragraphs are taken from [19].

In the formal grammatical rules for a language, each kind of syntactic unit or grouping is named by a *symbol*. Those which are built by grouping smaller constructs according to grammatical rules are called *nonterminal symbols*; those which can't be subdivided are called *terminal symbols* or *token types*. We call a piece of input corresponding to a single terminal symbol a *token*, and a piece corresponding to a single nonterminal symbol a *grouping*.

The *token type* is a terminal symbol defined in the grammar, such as `INTEGER`, `IDENTIFIER` or `','`. It tells everything you need to know to decide where the token may validly appear and how to group it with other tokens. The grammar rules know nothing about tokens except their types.

The *semantic value* has all the rest of the information about the meaning of the token, such as the value of an integer, or the name of an identifier. (A token such as `','` which is just punctuation doesn't need to have any semantic value.)

The job of the `yacc` parser is to group tokens into groupings according to the grammar rules, for example, to build identifiers and operators into expressions. As it does this, it runs the actions for the grammar rules it uses.

The tokens come from the lexical analyzer, which `yacc` parser calls each time it wants a new token. It doesn't know what is "inside" the tokens (though their semantic values may reflect this). Typically the lexical analyzer makes the tokens by parsing characters of text as already described in previous paragraphs.

The input file for `yacc` is a *yacc grammar file*. The general form of such a file has four main sections, shown here with the appropriate delimiters:

```
%{
C declarations
%}
Bison declarations
%%
Grammar rules
%%
Additional C code
```

The *C declarations* section contains macro definitions and declarations of functions and variables that are used in the actions in the grammar rules. `#include` can also be used to get the declarations from a header file.

The *declarations* section contains declarations that define terminal and nonterminal symbols, specify operator precedence and the data types of semantic values of various symbols. Operator precedence is determined by the line ordering of the declarations: the higher the line number of the declaration, the higher the precedence.

The *grammar rules* section contains one or more grammar rules that define how to construct each nonterminal symbol from its parts.

The *additional C code* can contain any C code we want to use. Often the definition of the lexical analyzer `yylex` goes here, plus subroutines called by the actions in the grammar rules. In a simple program, all the rest of the program can go here.

The `%union` declaration specifies the entire collection of possible data types for semantic values. For example:

```
%union {
  int intval;
  double dval;
  symrec *tptr;
}
```

This says that the three alternative types are `int,` `double` and `symrec *`. They are given names `intval,` `dval` and `tptr`; these names are used in the `%token` and `%type` declarations to pick one of the types for a terminal or nonterminal symbol.

The semantic value of a token returned by the lexical analyzer (if it has one) is stored into the global variable `yylval`, which is where the `yacc` parser will look for it. When we use multiple data types, `yylval`'s type is a union made from the `%union` declaration. So when we store a token's value, we must use the proper member of the union. So for the above `%union` declaration the code in `yylex` might look like this:

```
  yylval.intval = value; /* Put value onto yacc stack. */
  return INT;          /* Return the type of the token. */
```

A grammar rule has the following general form:

```
result: components...
        ;
```

where *result* is the nonterminal symbol that this rule describes, and *components* are various terminal and nonterminal symbols that are put together by this rule.

For example,

```
exp:       exp '+' exp
        ;
```

says that two groupings of type `exp`, with a + token in between, can be combined into a larger grouping of type `exp`.

An action accompanies a syntactic rule and contains C code to be executed each time an instance of that rule is recognized. An action looks like this:

```
{C statements}
```

The task of most actions is to compute a semantic value for the grouping built by the rule from the semantic values associated with tokens or smaller groupings.

The C code in an action can refer to the semantic values of the components matched by the rule with the construct `$`*n*, which stands for the value of the *n*th component. The semantic value for the grouping being constructed is `$$`.

Here is a typical example:

```
exp:      ...
        | exp '+' exp
        { $$ = $1 + $3; }
```

This rule constructs an exp from two smaller exp groupings connected by a plus-sign token. In the action, $1 and $3 refer to the semantic values of the two component exp groupings, which are the first and third symbols on the right hand side of the rule. The sum is stored into $$ so that it becomes the semantic value of the addition-expression just recognized by the rule. If there were a useful semantic value associated with the + token, it could be referred to as $2.

REMARK:
yacc has the option `-p prefix' which renames the external symbols used in the parser (including yyparse, yyerror, yylval), so that they start with *prefix* instead of yy. In our case we use `-p kd', so the names become kdparse, kdlval, and so on.

4.4 Using yacc in KeyNote

4.4.1 Building the Expression Tree

We need to *record* the expression described in the Conditions field of the KeyNote policy file, while reading it, in order to further process it. The best way to do this is by using a *tree* [20].

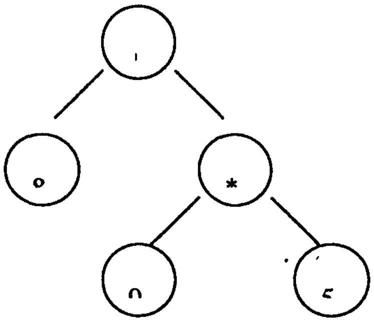
To understand how a tree works, consider an expression such as:

8 + 9 * 5

which is evaluated as:

8 + (9 * 5)

Each operation has these components: the operator and the operands.



A tree node may contain an operator. The node's branches (or its children) will represent the operands of the operator. The end branches that are simple operands, not expressions, are called *leaves*.

Tree structures are a good way to represent logical expressions as well. They record all the information needed to evaluate the expression.

To represent the tree of the logical expression we need the data types below (based on examples from [20]), which are defined in the header file tree.h:

```
struct node
{
       char* operator;
       union
       {
              char* value;
              struct node *np;
       } left, right;
};
#define LCHILD(snode)       ((snode)->left.np)
#define RCHILD(snode)       ((snode)->right.np)
#define NNULL        ((struct node *) 0)
```

There can be two types of nodes: intermediate and leaf/terminal nodes.

An intermediate node uses the fields:
`operator`, which can be: a logical or relational operator
`left`, which contains:
`np` pointer to left child node
and may also use (if the operator is binary)
`right`, which contains:
`np` pointer to right child node

A leaf node has:
`operator`, which can be: a `VARIABLE` or `STRING` indicator
(for a KeyNote attribute or an attribute value respectively)
`left`, which contains:
`value` semantic value of `VARIABLE` or `STRING`

The fields not mentioned for an intermediate or leaf node are empty or ignored.

The `yacc` file of our utility can be found in Appendix B. `yacc` was already used in KeyNote code for processing of KeyNote files. The grammar in our `yacc` program is based on the grammar defined in the `keynote.y` [21] and a description of the key points in it follows.

To record an expression, we use `nalloc()` to allocate a `node` structure.

The values returned by actions and declared in the `%union` definition are: a pointer to a string, an integer or a pointer to a node structure.

The tokens and nonterminal symbols defined are a subset of those in `keynote.y`.

In the rules section the bare grammar without actions attached to the rules in the rules section is a subset of that in `keynote.y`. The rules that do not refer to the Conditions field of KeyNote policy file are omitted, as well as those dealing with numerical and float expressions.

The actions assigned to the rest of the rules are modified to allow the building of the expression tree as outlined below:

- The rules of our grammar add leaves to the tree, when we meet one of the tokens:
  - VARIABLE

    A KeyNote attribute was returned by `lex`, and the function `add_leaf()` fills in the operator field of the `leaf` node with the `VARIABLE` indicator and the `left` field with the actual name of the KeyNote attribute
  - STRING

    A KeyNote attribute value was returned by `lex`, and the function `add_leaf()` fills in the `operator` field of the `leaf` node with the `STRING` indicator and the `left` field with the actual string representing the KeyNote attribute's value
  - TRUE/FALSE,

These tokens where found by `lex` in the KeyNote file, and the function `add_leaf()` fills in the `operator` field of the leaf node with the STRING indicator and the `left` field with the actual token string.

- The rules of our grammar add nodes to the tree, when logical and relational expressions are met, like:

```
expr AND expr
expr OR expr
NOT expr
str EQ str
str NE str
str LT str
str GT str
str LE str
str GE str
```

In this case the function `add_node()` fills in

- the `operator` field of the node with one of the indicators &&,||,!,==,!=,<,>,<=,>=
- the `left` field with the pointer to the subtree of the left expression
- the `right` field, if the operator is binary, with the pointer to the subtree of the right expression

As input is collected, the tree structure is allocated and organized. When the complete statement has been collected, we are ready to convert it to DNF.

4.4.2 Converting the Expression Tree to DNF

When we reach the rule

```
program: prog
```

the tree representing the Boolean expression described in the Conditions field of KeyNote policy file has been built and we can process it to convert it to DNF.

The first step is to move all the negation signs in to the atoms by applying de Morgan laws. While doing this we eliminate double negations. The function doing this is `permeate_nots`, which accepts as input the root node of a tree/subtree and uses recursion. We'll briefly describe its functionality using example trees:

When the root node is a NOT operator, it only has one child. The child node is examined and in case it is:

1) a logic operator and more specifically:

a) a NOT operator

The tree of the left side of Figure 3 is converted to the tree in the right side of Figure 3, by eliminating nodes (1) and (2) and the function is called recursively for node (3).

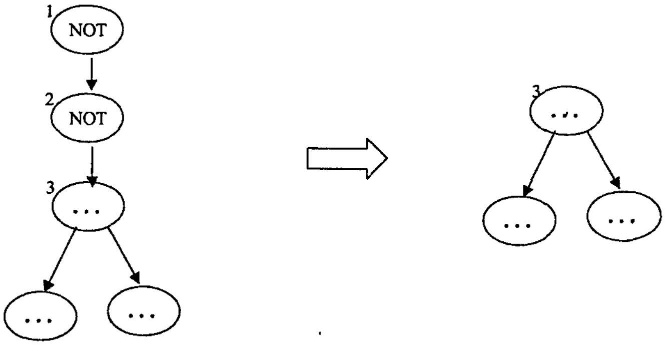


Figure 3

b) an AND operator

The tree of the left side of Figure 4 is converted to the tree in the right side of Figure 4.

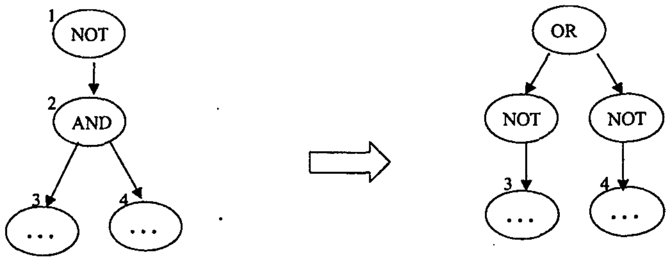


Figure 4

c) an OR operator

The tree of the left side of Figure 5 is converted to the tree in the right side of Figure 5.

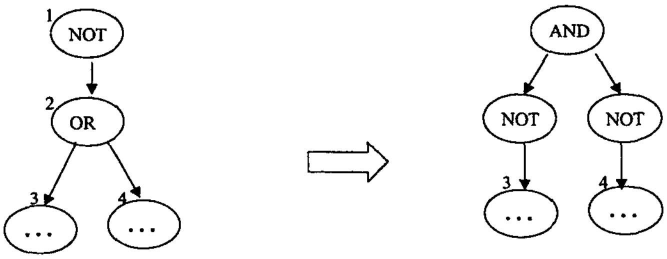


Figure 5

2) a relational operator

The tree of the left side of Figure 6 is converted to the tree in the right side of Figure 6.

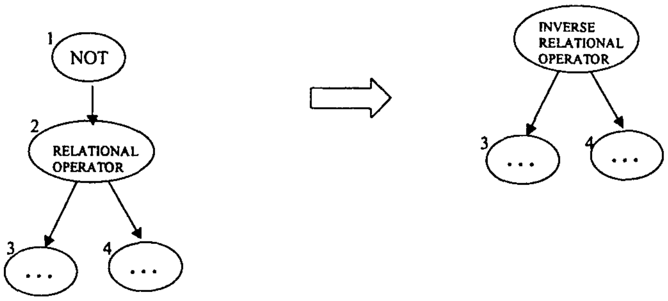


Figure 6

The relational operator is inverted according to the table below:

RELATIONAL OPERATOR	INVERSE RELATIONAL OPERATOR
==	!=
!=	==
<	>=
>	<=
<=	>
>=	<

At the end of the function if the root node's children are not leaves, the function is applied recursively to them.

The <u>second step</u> is the use of the distributive identity to create a disjunction of conjunctions.

The function doing this is `and_distribute`, which accepts as input the root node of a tree/subtree and uses recursion. We'll briefly describe its functionality using example trees:

The root node is examined. In case it is

1) an OR operator

The function is called recursively for the node's children.

2) an AND operator

The right child is examined and if it is an OR, the tree of the left side of Figure 7 is converted to the tree in the right side of Figure 7.

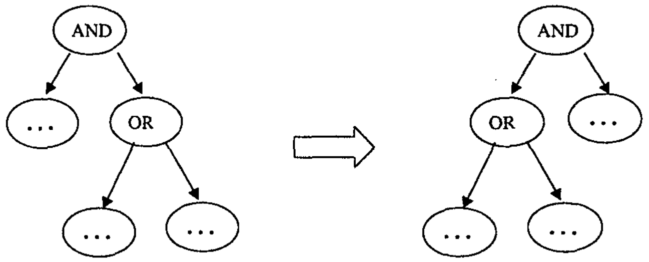


Figure 7

Then the left child is examined. In case it is:

a) an OR operator,

the tree of the left side of Figure 8 is converted to the tree in the right side of Figure 8, and the function is called recursively for the OR node of the right tree.

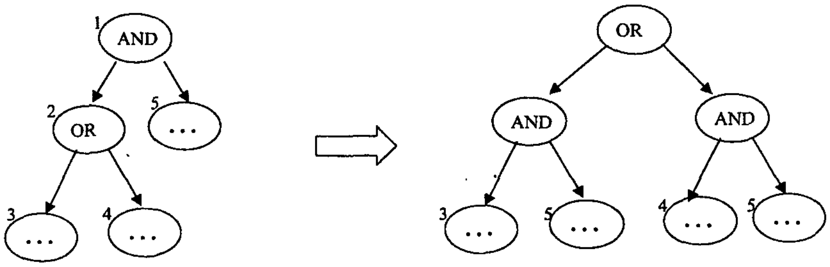


Figure 8

b) if it is not an OR operator, the function is called recursively for its children, if they are not leaf nodes.

This procedure is repeated until there are no more changes due to the distributive law on the tree.

The last step is writing the tree -which is now in DNF- to a file (the file name was accepted as input in the function `convert_to_dnf()` that activated `lex` and `yacc` parsing) by calling `print_tree_file()`. When this operation is completed, the tree is no longer needed, so the function named `free_tree()` is called to free the memory used for all the structures that make up the tree.

For the conversion to DNF we used some code from the Managing Gigabytes (MG) project, which is an open-source indexing and retrieval system for text, images, and textual images [22]. MG is covered by a GNU public license. The current version of MG is available for ftp from http://www.cs.mu.oz.au/mg/.

More specifically the files below were consulted:

```
bool_optimizer.c, bool_optimizer.h,
bool_parser.c, bool_parser.h, bool parser.y
bool_tester.c
bool_tree.c, bool_tree.h
term_lists.c, term_lists.h
```

words.h

NOTE: Our code does not attempt to simplify the expression e.g. check for the presence of both a term and its negation in a conjunction and remove it or check whether a term appears twice in a conjunction. These issues can be addressed in the future.

5. Example of input KeyNote policy file and generated DNF

An example of an input isakmpd.policy KeyNote policy file containing our QoSS attributes can be seen below:

```
KeyNote-Version: 2
Comment: Policy file for Network Modes and Security Levels
Authorizer: "POLICY"
Licensees: "passphrase:mekmitasdigoat"
Conditions: ( (app_domain == "IPsec policy") &&
              (
                ( (network_mode == "normal") &&
                  (
                    ( (security_level == "low") &&
                      (
                        ( (esp_present == "yes") &&
                          ( (local_filter_port == "23") || (remote_filter_port == "23") ) &&
                          (esp_enc_alg == "des") &&
                          (esp_auth_alg == "hmac-md5")
                        )
                        ||
                        ( (ah_present == "yes") &&
                          ( (local_filter_port == "79") || (remote_filter_port == "79") ) &&
                          (ah_auth_alg == "hmac-md5")
                        )
                      )
                    )
                    ||
                    ( (security_level == "medium") &&
                      (
                        ( (esp_present == "yes") &&
                          ( (local_filter_port == "23") || (remote_filter_port == "23") ) &&
                          (esp_enc_alg == "cast") &&
                          (esp_auth_alg == "hmac-sha")
                        )
                        ||
                        ( (ah_present == "yes") &&
                          ( (local_filter_port == "79") || (remote_filter_port == "79") ) &&
                          (ah_auth_alg == "hmac-md5")
                        )
                      )
                    )
                    ||
                    ( (security_level == "high") &&
                      (
                        ( (esp_present == "yes") &&
                          ( (local_filter_port == "23") || (remote_filter_port == "23") ) &&
                          (esp_enc_alg == "3des") &&
                          (esp_auth_alg == "hmac-sha")
                        )
                        ||
                        ( (ah_present == "yes") &&
                          ( (local_filter_port == "79") || (remote_filter_port == "79") ) &&
                          (ah_auth_alg == "hmac-sha")
                        )
                      )
                    )
                  )
                )
                ||
                ( (network_mode == "impacted") &&
  (
    ( (security_level == "low") &&
      (
        ( (esp_present == "yes") &&
          ( (local_filter_port == "23") || (remote_filter_port == "23") ) &&
          (esp_enc_alg == "des") &&
          (esp_auth_alg == "hmac-md5")
        )
        ||
        ( (ah_present == "yes") &&
          ( (local_filter_port == "79") || (remote_filter_port == "79") ) &&
          (ah_auth_alg == "hmac-md5")
        )
      )
    )
    ||
    ( (security_level == "medium") &&
      (
        ( (esp_present == "yes") &&
          ( (local_filter_port == "23") || (remote_filter_port == "23") ) &&
          (esp_enc_alg == "des") &&
          (esp_auth_alg == "hmac-md5")
        )
        ||
        ( (ah_present == "yes") &&
          ( (local_filter_port == "79") || (remote_filter_port == "79") ) &&
          (ah_auth_alg == "hmac-md5")
        )
      )
    )
    ||
    ( (security_level == "high") &&
      (
        ( (esp_present == "yes") &&
          ( (local_filter_port == "23") || (remote_filter_port == "23") ) &&
          (esp_enc_alg == "3des") &&
          (esp_auth_alg == "hmac-md5")
        )
        ||
        ( (ah_present == "yes") &&
          ( (local_filter_port == "79") || (remote_filter_port == "79") ) &&
          (ah_auth_alg == "hmac-sha")
        )
      )
    )
  )
)
||
( (network_mode == "crisis") &&
  (
    ( (security_level == "low") &&
      (
        ( (esp_present == "yes") &&
          ( (local_filter_port == "23") || (remote_filter_port == "23") ) &&
          (esp_enc_alg == "3des") &&
          (esp_auth_alg == "hmac-sha")
        )
        ||
        ( (ah_present == "yes") &&
          ( (local_filter_port == "79") || (remote_filter_port == "79") ) &&
          (ah_auth_alg == "hmac-sha")
        )
      )
    )
    ||
    ( (security_level == "medium") &&
      (
        ( (esp_present == "yes") &&
          ( (local_filter_port == "23") || (remote_filter_port == "23") ) &&
          (esp_enc_alg == "3des") &&
          (esp_auth_alg == "hmac-sha")
```

```
                        )
                        ||
                        ( (ah_present == "yes") &&
                          ( (local_filter_port == "79") || (remote_filter_port == "79") ) &&
                          (ah_auth_alg == "hmac-sha")
                        )
                      )
                    )
                    ||
                    ( (security_level == "high") &&
                      (
                        ( (esp_present == "yes") &&
                          ( (local_filter_port == "23") || (remote_filter_port == "23") ) &&
                          (esp_enc_alg == "aes") &&
                          (esp_auth_alg == "hmac-sha")
                        )
                        ||
                        ( (ah_present == "yes") &&
                          ( (local_filter_port == "79") || (remote_filter_port == "79") ) &&
                          (ah_auth_alg == "hmac-sha")
                        )
                      )
                    )
                  )
                )
                ||
                ( (network_mode == "default") &&
                  (security_level == "default") &&
                  (
                    ( (esp_present == "yes") &&
                      ( (local_filter_port == "23") || (remote_filter_port == "23") ) &&
                      (esp_enc_alg == "des") &&
                      (esp_auth_alg == "hmac-md5")
                    )
                    ||
                    ( (ah_present == "yes") &&
                      ( (local_filter_port == "79") || (remote_filter_port == "79") ) &&
                      (ah_auth_alg == "hmac-md5")
                    )
                  )
                )
              )
            )
     -> "true";
```

The output of our utility can be seen below, which is the DNF of the Conditions field of the input file:

```
((((((((((((local_filter_port == "23") && (esp_present == "yes")) &&
(esp_enc_alg == "des")) && (esp_auth_alg == "hmac-md5")) &&
(security_level == "low")) && (network_mode == "normal")) &&
(app_domain == "IPsec policy"))
||
((((((remote_filter_port == "23") && (esp_present == "yes")) &&
(esp_enc_alg == "des")) && (esp_auth_alg == "hmac-md5")) &&
(security_level == "low")) && (network_mode == "normal")) &&
(app_domain == "IPsec policy")))
||
((((((local_filter_port == "79") && (ah_present == "yes")) &&
(ah_auth_alg == "hmac-md5")) && (security_level == "low")) &&
(network_mode == "normal")) && (app_domain == "IPsec policy"))
||
(((((remote_filter_port == "79") && (ah_present == "yes")) &&
(ah_auth_alg == "hmac-md5")) && (security_level == "low")) &&
(network_mode == "normal")) && (app_domain == "IPsec policy"))))
||
((((((((local_filter_port == "23") && (esp_present == "yes")) &&
(esp_enc_alg == "cast")) && (esp_auth_alg == "hmac-sha")) &&
(security_level == "medium")) && (network_mode == "normal")) &&
(app_domain == "IPsec policy"))
||
(((((((remote_filter_port == "23") && (esp_present == "yes")) &&
(esp_enc_alg == "cast")) && (esp_auth_alg == "hmac-sha")) &&
(security_level == "medium")) && (network_mode == "normal")) &&
(app_domain == "IPsec policy")))
||
(((((((local_filter_port == "79") && (ah_present == "yes")) &&
(ah_auth_alg == "hmac-md5")) && (security_level == "medium")) &&
(network_mode == "normal")) && (app_domain == "IPsec policy"))
||
((((((remote_filter_port == "79") && (ah_present == "yes")) &&
(ah_auth_alg == "hmac-md5")) && (security_level == "medium")) &&
(network_mode == "normal")) && (app_domain == "IPsec policy")))))
||
(((((((((local_filter_port == "23") && (esp_present == "yes")) &&
(esp_enc_alg == "3des")) && (esp_auth_alg == "hmac-sha")) &&
(security_level == "high")) && (network_mode == "normal")) &&
(app_domain == "IPsec policy"))
||
(((((((remote_filter_port == "23") && (esp_present == "yes")) &&
(esp_enc_alg == "3des")) && (esp_auth_alg == "hmac-sha")) &&
(security_level == "high")) && (network_mode == "normal")) &&
(app_domain == "IPsec policy")))
||
(((((((local_filter_port == "79") && (ah_present == "yes")) &&
(ah_auth_alg == "hmac-sha")) && (security_level == "high")) &&
(network_mode == "normal")) && (app_domain == "IPsec policy"))
||
((((((remote_filter_port == "79") && (ah_present == "yes")) &&
(ah_auth_alg == "hmac-sha")) && (security_level == "high")) &&
(network_mode == "normal")) && (app_domain == "IPsec policy")))))
||
(((((((((((local_filter_port == "23") && (esp_present == "yes")) &&
(esp_enc_alg == "des")) && (esp_auth_alg == "hmac-md5")) &&
(security_level == "low")) && (network_mode == "impacted")) &&
(app_domain == "IPsec policy"))
||
(((((((remote_filter_port == "23") && (esp_present == "yes")) &&
(esp_enc_alg == "des")) && (esp_auth_alg == "hmac-md5")) &&
(security_level == "low")) && (network_mode == "impacted")) &&
(app_domain == "IPsec policy")))
||
(((((((local_filter_port == "79") && (ah_present == "yes")) &&
(ah_auth_alg == "hmac-md5")) && (security_level == "low")) &&
(network_mode == "impacted")) && (app_domain == "IPsec policy"))
||
((((((remote_filter_port == "79") && (ah_present == "yes")) &&
(ah_auth_alg == "hmac-md5")) && (security_level == "low")) &&
(network_mode == "impacted")) && (app_domain == "IPsec policy"))))
||
(((((((((local_filter_port == "23") && (esp_present == "yes")) &&
(esp_enc_alg == "des")) && (esp_auth_alg == "hmac-md5")) &&
(security_level == "medium")) && (network_mode == "impacted")) &&
(app_domain == "IPsec policy"))
||
(((((((remote_filter_port == "23") && (esp_present == "yes")) &&
(esp_enc_alg == "des")) && (esp_auth_alg == "hmac-md5")) &&
(security_level == "medium")) && (network_mode == "impacted")) &&
(app_domain == "IPsec policy")))
||
(((((((local_filter_port == "79") && (ah_present == "yes")) &&
(ah_auth_alg == "hmac-md5")) && (security_level == "medium")) &&
(network_mode == "impacted")) && (app_domain == "IPsec policy"))
||
((((((remote_filter_port == "79") && (ah_present == "yes")) &&
(ah_auth_alg == "hmac-md5")) && (security level == "medium")) &&
(network_mode == "impacted")) && (app_domain == "IPsec policy")))))
||
(((((((((local_filter_port == "23") && (esp_present == "yes")) &&
(esp_enc_alg == "3des")) && (esp_auth_alg == "hmac-md5")) &&
(security_level == "high")) && (network_mode == "impacted")) &&
(app_domain == "IPsec policy"))
||
(((((((remote_filter_port == "23") && (esp_present == "yes")) &&
(esp_enc_alg == "3des")) && (esp_auth_alg == "hmac-md5")) &&
(security_level == "high")) && (network_mode == "impacted")) &&
(app_domain == "IPsec policy")))
||
(((((((local_filter_port == "79") && (ah_present == "yes")) &&
(ah_auth_alg == "hmac-sha")) && (security_level == "high")) &&
(network_mode == "impacted")) && (app_domain == "IPsec policy"))
||
((((((remote_filter_port == "79") && (ah_present == "yes")) &&
(ah_auth_alg == "hmac-sha")) && (security_level == "high")) &&
(network_mode == "impacted")) && (app_domain == "IPsec policy"))))))
||
(((((((((((local_filter_port .== "23") && (esp_present == "yes")) &&
(esp_enc_alg == "3des")) && (esp_auth_alg == "hmac-sha")) &&
(security_level == "low")) && (network_mode == "crisis")) &&
(app_domain == "IPsec policy"))
||
(((((((remote_filter_port == "23") && (esp_present == "yes")) &&
(esp_enc_alg == "3des")) && (esp_auth_alg == "hmac-sha")) &&
(security_level == "low")) && (network_mode == "crisis")) &&
(app_domain == "IPsec policy")))
||
(((((((local_filter_port == "79") && (ah_present == "yes")) &&
(ah_auth_alg == "hmac-sha")) && (security_level == "low")) &&
(network_mode == "crisis")) && (app_domain == "IPsec policy"))
||
((((((remote_filter_port == "79") && (ah_present == "yes")) &&
(ah_auth_alg == "hmac-sha")) && (security_level == "low")) &&
(network_mode == "crisis")) && (app_domain == "IPsec policy"))))
||
(((((((((local_filter_port == "23") && (esp_present == "yes")) &&
(esp_enc_alg == "3des")) && (esp_auth_alg == "hmac-sha")) &&
(security_level == "medium")) && (network_mode == "crisis")) &&
(app_domain == "IPsec policy"))
||
(((((((remote_filter_port == "23") && (esp_present == "yes")) &&
(esp_enc_alg == "3des")) && (esp_auth_alg == "hmac-sha")) &&
(security_level == "medium")) && (network_mode == "crisis")) &&
(app_domain == "IPsec policy")))
||
(((((((local_filter_port == "79") && (ah_present == "yes")) &&
(ah_auth_alg == "hmac-sha")) && (security_level == "medium")) &&
(network_mode == "crisis")) && (app_domain == "IPsec policy"))
||
((((((remote_filter_port == "79") && (ah_present == "yes")) &&
(ah_auth_alg == "hmac-sha")) && (security_level == "medium")) &&
(network_mode == "crisis")) && (app_domain == "IPsec policy")))))
||
(((((((((local_filter_port == "23") && (esp_present == "yes")) &&
(esp_enc_alg == "aes")) && (esp_auth_alg == "hmac-sha")) &&
(security_level == "high")) && (network_mode == "crisis")) &&
(app_domain == "IPsec policy"))
||
(((((((remote_filter_port == "23") && (esp_present == "yes")) &&
(esp_enc_alg == "aes")) && (esp_auth_alg == "hmac-sha")) &&
(security_level == "high")) && (network_mode == "crisis")) &&
(app_domain == "IPsec policy")))
||
(((((((local_filter_port == "79") && (ah_present == "yes")) &&
(ah_auth_alg == "hmac-sha")) && (security_level == "high")) &&
(network_mode == "crisis")) && (app_domain == "IPsec policy"))
||
((((((remote_filter_port == "79") && (ah_present == "yes")) &&
(ah_auth_alg == "hmac-sha")) && (security_level == "high")) &&
(network_mode == "crisis")) && (app_domain == "IPsec policy"))))))
||
((((((((local_filter_port == "23") && (esp_present == "yes")) &&
(esp_enc_alg == "des")) && (esp_auth_alg == "hmac-md5")) &&
((network_mode == "default") && (security_level == "default"))) &&
(app_domain == "IPsec policy"))
||
(((((remote_filter_port == "23") && (esp_present == "yes")) &&
(esp_enc_alg == "des")) && (esp_auth_alg == "hmac-md5")) &&
((network_mode == "default") && (security_level == "default"))) &&
(app_domain == "IPsec policy")))
||
(((((local_filter_port == "79") && (ah_present == "yes")) &&
(ah_auth_alg == "hmac-md5")) && ((network_mode == "default") &&
(security_level == "default"))) && (app_domain == "IPsec policy"))
||
((((remote_filter_port == "79") && (ah_present == "yes")) &&
 (ah_auth_alg == "hmac-md5")) && ((network_mode == "default") &&
 (security_level == "default"))) && (app_domain == "IPsec policy")))))
```

6. References

[1] Blaze, M., Feigenbaum, J., Ioannidis, J. and Keromytis, A.D., "The KeyNote Trust Management System Version 2", Internet RFC 2704, Internet Engineering Task Force, September 1999.

[2] keynote(4), OpenBSD Programmer's Manual, http://www.openbsd.org/cg-bin/man.cgi, May 1999.

[3] keynote(5), OpenBSD Programmer's Manual, http://www.openbsd.org/cg-bin/man.cgi, October 1999.

[4] Kent. S. and Atkinson, R., "Security Architecture for the Internet Protocol", Internet RFC 2401, Internet Engineering Task Force, November 1998.

[5] Blaze, M., Ioannidis, J. and Keromytis, A.D., "Trust Management for IPSec", Proc. of the Internet Society Symposium on Network and Distributed Systems Security 2001, San Diego, CA, February 2001, pp. 139-151.

[6] isakmpd.policy(5), OpenBSD Programmer's Manual, http://www.openbsd.org/cg-bin/man.cgi, October 1998.

[7] Fricke, T., Boolean Normal Forms, http://splorg.org/~tobin/projects/quinto/dnf.html, August 2001.

[8] Weisstein E.W., Disjunctive Normal Form – from Mathworld, http://mathworld.wolfram.com/DisjunctiveNormalForm.html.

[9] Garrett Birkhoff, Lattice Theory, American Mathematical Society, Providence, Rhode Island, 1967, pp. 61-63.

[10] Irvine, C. and Levin, T., "Quality of Security Service", Proc. of New Security Paradigms Workshop 2000, Cork, Ireland, September 2000, pp. 91-99.

[11] Irvine, C. and Levin, T., "A Note on Mapping User-Oriented Security Policies to Complex Mechanisms and Services", Technical Report NPS-CS-99-08, Naval Postgraduate School, Monterey, CA, June 1999.

[12] Spyropoulou, E., Agar, C., Levin, T., and Irvine, C., "IPsec Modulation for Quality of Security Service", Technical Report NPS-CS-02-01, Naval Postgraduate School, Monterey, CA, January 2002.

[13] Spyropoulou, E., Levin, T., and Irvine, C., "Demonstration of Quality of Security Service Awareness for IPsec", Technical Report NPS-CS-02-03, Naval Postgraduate School, Monterey, CA, January 2002.

[14] Agar, C.D., "Dynamic Parameterization of IPsec", Master Thesis, Naval Postgraduate School, Monterey, CA, December 2001.

[15] Johnson S.C., "Yacc: Yet Another Compiler-Compiler", Bell Laboratories, Murray Hill, New Jersey, July 1978.

[16] Paxson V., "Flex, version 2.5. A fast scanner generator.", Edition 2.5, http://www.gnu.org/manual/flex-2.5.4/html_mono/flex.html, March 1995.

[17] `keynote.l`, CVS Repository, http://www.openbsd.org/cgi-bin/cvsweb/src/lib/libkeynote/keynote.l, August 2000.

[18] `keynote-ver.l`, CVS Repository, http://www.openbsd.org/cgi-bin/cvsweb/src/lib/libkeynote/keynote-ver.l, August 2000.

[19] Donelly, C. and Stallman, R., "Bison. The YACC-compatible Parser Generator", Bison Version 1.25, http://www.gnu.org/manual/bison/html_mono/bison.html, November 1995.

[20] "OS/390 V2R10.0 UNIX System Services Programming Tools" via IBM BookManager BookServer, http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/BPXA6071/1.6?DT=20001016170315, Document Number: SC28-1904-08, October 2000.

[21] `keynote.y`, CVS Repository, http://www.openbsd.org/cgi-bin/cvsweb/src/lib/libkeynote/keynote.y, October 2000.

[22] Managing Gigabytes, Compressing and Indexing Documents and Images, Second Edition, http://www.cs.mu.oz.au/mg/, August 1999.

APPENDIX A

lex Code File: keynote-dnf.l

```
%{
#include "tree.h"
#include "d.tab.h"
#include "assertion.h"

static void mystrncpy(char *, char *, int);
static unsigned char get_octal(char *, int, int *);
static int is_octal(char);

static int    first_tok = 0;
%}
digit         [0-9]
specnumber       [1-9][0-9]*
number        {digit}+
flt           {digit}+"."{digit}+
vstring              [a-zA-Z_][a-zA-Z0-9_]*
litstring     \"(((\\\n)|(\\.)|([^\\\n\"]))*)\"
variable      {vstring}
comment              "#"[^\n]*
%s OUTCOND CONDITIONS
%pointer
%option noyywrap never-interactive yylineno
%%
%{
    /*
     * Return a preset token, so we can have more than one grammars
     * in yacc.
     *
     * REMARK: taken as is from keynote.l
     */
    extern int first_tok;

    if (first_tok)
    {
       int holdtok = first_tok;

       first_tok = 0;
       return holdtok;
    }
%}
<OUTCOND>"Conditions"       {
                           BEGIN(CONDITIONS);
                     }
<CONDITIONS>"("    {
                     return OPENPAREN;
                   }
<CONDITIONS>")"    {
                     return CLOSEPAREN;
                   }
<CONDITIONS>"&&"   {
                     return AND;
                   }
<CONDITIONS>"||"   {
                     return OR;
                   }
<CONDITIONS>"->"   {
```

```
                    return HINT;
                  }
<CONDITIONS>"{"   {
                    return OPENBLOCK;
                  }
<CONDITIONS>"}"   {
                    return CLOSEBLOCK;
                  }
<CONDITIONS>";"   {
                    return SEMICOLON;
                  }
<CONDITIONS>"!"   {
                    return NOT;
                  }
<CONDITIONS>"~="  {
                    return REGEXP;
                  }
<CONDITIONS>"=="  {
                    return EQ;
                  }
<CONDITIONS>"!="  {
                    return NE;
                  }
<CONDITIONS>"<"   {
                    return LT;
                  }
<CONDITIONS>">"   {
                    return GT;
                  }
<CONDITIONS>"<="  {
                    return LE;
                  }
<CONDITIONS>">="  {
                    return GE;
                  }
<CONDITIONS>"."   {
                    return DOTT;
                  }
<CONDITIONS>"KeyNote-Version"
                  {
                    BEGIN(OUTCOND);
                  }
<CONDITIONS>"Comment"
                  {
                    BEGIN(OUTCOND);
                  }
<CONDITIONS>"Authorizer"
                  {
                    BEGIN(OUTCOND);
                  }
<CONDITIONS>"Licensees"
                  {
                    BEGIN(OUTCOND);
                  }
<CONDITIONS>"Signature"
                  {
                    BEGIN(OUTCOND);
                  }
<CONDITIONS>"true"  {
                      return TRUE;
                    }
<CONDITIONS>"false" {
                      return FALSE;
                    }
{comment}           /* eat up comments */
<CONDITIONS>{variable} {
                    /* XXX keynote_exceptionflag, keynote_donteval issues
                    have not been addressed, so the code below from
                    keynote.l is commented out */
                    /*
                    if (keynote_exceptionflag ||
                            keynote_donteval)
                    {
                      kdlval.string = (char *) NULL;
                      return VARIABLE;
                    } */

                    kdlval.string = calloc(strlen(kdtext) + 1,
                                           sizeof(char));
                    if (kdlval.string == (char *) NULL)
                    {
                      /* XXX keynote_errno issues have not been addressed,
                      so the code below from keynote.l is commented out */
                      /*keynote_errno = ERROR_MEMORY;*/
                      return -1;
                    }
                    strcpy(kdlval.string, kdtext);
                    return VARIABLE;
                  }
<CONDITIONS>"$"   {
                    return DEREF;
                  }
<CONDITIONS>{litstring}   {
                    /* XXX keynote_exceptionflag, keynote_donteval issues
                    have not been addressed, so the code below from
                    keynote.l is commented out */
                    /*if (keynote_exceptionflag ||
                            keynote_donteval)
                    {
                      kdlval.string = (char *) NULL;
                      return STRING;
                    } */

                    kdlval.string = calloc(strlen(kdtext) - 1,
                                    sizeof(char));
                    if (kdlval.string == (char *) NULL)
                    {
                      /* XXX keynote_errno issues have not been addressed,
                      so the code below from keynote.l is commented out */
                      /*keynote_errno = ERROR_MEMORY;*/
                      return -1;
                    }

                    mystrncpy(kdlval.string, kdtext + 1,
                              strlen(kdtext) - 2);
                    return STRING;
                  }
[ \t\n]
                  {
                    /* XXX keynote_errno issues and lex/yacc return values
                    have not been addressed systematically,
                    so the code below from keynote.l is commented out */
                    /*keynote_errno = ERROR_SYNTAX;
                    return -1;
                    REJECT; */ /* Avoid -Wall warning. Not reached */
                  }
```

```
%%

/******************************************************************
 * int convert_to_dnf(char *policyfile, char *dnffile)
 * This is the function that activates lex and yacc parsing
 * through the call to kdparse()
 * It creates a buffer containing the input file for the parser
 * and calls the parser for it. It copies the ouput file name to
 * the global variable for the file name in which yacc writes its
 * output.
 * XXX Incomplete handling of error flags, error numbers and
 * return value...
 * Inputs:
 *     char *policyfile -  name of file to be converted to DNF
 *     char *dnffile    -  name of file where DNF will be stored
 ******************************************************************/
int convert_to_dnf(char *policyfile, char *dnffile)
{
  YY_BUFFER_STATE kdbuffer;
  FILE *fp;
  int i;

  targetdnf = dnffile;
  fp = fopen(policyfile, "r");
  if (fp == (FILE *) NULL)
  {
    perror(policyfile);
    return -1;
  }

  kdbuffer = kd_create_buffer(fp, YY_BUF_SIZE);
  kd_switch_to_buffer(kdbuffer);
  BEGIN(OUTCOND);
  first_tok = ACTSTR;
  i = kdparse();
  kd_delete_buffer(kdbuffer);
  fclose(fp);
  fprintf(stderr, "convert_to_dnf: FINISHED, result = %i \n", i);
  return -99888; /* XXX return an arbitrary value for test purposes */

  /* XXX keynote_errno issues and lex/yacc return values
  have not been addressed systematically,
  so the code below from keynote-ver.1 is commented out */
/*  switch (i)
  {
    case 0:
      return 0;

    default:
      if (keynote_errno == ERROR_MEMORY)
        fprintf(stderr,
                "Memory error while processing policy file <%s>\n",
                policyfile);
      else
        fprintf(stderr,
                "Syntax error in environment file <%s>, line %d\n",
                policyfile, kdlineno);
      return -1;
    } */
}


/*
```

```
 * Copy at most len characters to string s1 from string s2, taking
 * care of escaped characters in the process. String s1 is assumed
 * to have enough space, and be zero'ed.
 *
 * REMARK: This function was copied from keynote.l. It also exists
 * identical in keynote-ver.l, so it seemed best to also copy it
 * here. Maybe there's a better way for doing this, instead of
 * keeping a seperate function in every lex file.
 */
static void
mystrncpy(char *s1, char *s2, int len)
{
  unsigned char c;
  int advance;

  if (len == 0)
    return;

  while (len-- > 0)
  {
    if (*s2 == '\\')
    {
      s2++;

      if (len-- <= 0)
        break;

      if (*s2 == '\n')
      {
        while (isspace((int) *(++s2)) && (len-- > 0))
          ;
      }
      else
        if ((c = get_octal(s2, len, &advance)) != 0)
        {
          len -= advance - 1;
          s2 += advance;
          *s1++ = c;
        }
        else
          if (*s2 == 'n')  /* Newline */
          {
            *s1++ = '\n';
            s2++;
          }
          else
            if (*s2 == 't')  /* Tab */
            {
              *s1++ = '\t';
              s2++;
            }
            else
              if (*s2 == 'r')  /* Linefeed */
              {
                *s1++ = '\r';
                s2++;
              }
              else
                if (*s2 == 'f')  /* Formfeed */
                {
                  *s1++ = '\f';
                  s2++;
                }
                else
                  if ((*s1++ = *s2++) == 0)
                    break;

    continue;
  }

  if ((*s1++ = *s2++) == 0)
    break;
  }
}

/*
 * Return octal value (non-zero) if argument starts with such a
 * representation, otherwise 0.
 *
 * REMARK: This function was copied from keynote.l. It also exists
 * identical in keynote-ver.l, so it seemed best to also copy it
 * here. Maybe there's a better way for doing this, instead of
 * keeping a seperate function in every lex file.
 */
static unsigned char
get_octal(char *s, int len, int *adv)
{
  unsigned char res = 0;

  if (*s == '0')
  {
    if (len > 0)
    {
      if (is_octal(*(s + 1)))
      {
        res = *(s + 1) - '0';
        *adv = 2;

        if (is_octal(*(s + 2)) && (len - 1 > 0))
        {
          res = res * 8 + (*(s + 2) - '0');
          *adv = 3;
        }
      }
    }
  }
  else
    if (is_octal(*s) && (len - 1 > 0))  /* Non-zero leading */
    {
      if (is_octal(*(s + 1)) &&
          is_octal(*(s + 2)))
      {
        *adv = 3;
        res = (((*s) - '0') * 64) +
              (((*(s + 1)) - '0') * 8) +
              ((*(s + 2)) - '0');
      }
    }

  return res;
}

/*
 * Return RESULT_TRUE if character is octal digit, RESULT_FALSE otherwise.
 *
 * REMARK: This function was copied from keynote.l. It also exists
 * identical in keynote-ver.l, so it seemed best to also copy it
 * here. Maybe there's a better way for doing this, instead of
 * keeping a seperate function in every lex file.
 */
static int
is_octal(char c)
{
  switch (c)
  {
    case '0': case '1': case '2': case '3':
    case '4': case '5': case '6': case '7':
      return 1; /*RESULT_TRUE;*/
    default:
      return 0; /*RESULT_FALSE;*/
  }
}
```

APPENDIX B

yacc Code File: **keynote-dnf.y**

```
%{
#include "assertion.h"
#include "tree.h"
#include <stdio.h>
#include <errno.h>

struct node* nalloc();
struct node* add_leaf(char*, char*);
struct node* add_node(char*, struct node*, struct node*);
int  binary_node(struct node*);
void  print_binary_node(struct node*);
void  print_unary_node(struct node*);
void  print_tree(struct node*);
void  fprint_binary_node(FILE* file, struct node*);
void  fprint_unary_node(FILE* file, struct node*);
void  fprint_tree(FILE* file, struct node*);
void  print_tree_file(struct node*);
void  free_tree(struct node*);
void  apply_double_negation(struct node*);
void  apply_and_DeMorgan(struct node*);
void  apply_or_DeMorgan(struct node*);
void  inverse_tokens(struct node*);
void  permeate_nots(struct node*);
struct node* copy_bool_tree(struct node*);
void  distribute_branches(struct node*);
int   and_distribute(struct node*);
void  distribute_DNF(struct node*);
void  extract_DNF_tree(struct node*);

static int   keynote_donteval, keynote_exceptionflag;
%}
%union {
  char   *string;
  int     intval;
  struct node   *np;
};
%type <intval> afterhint
%type <string> STRING VARIABLE
%type <np> strnotconcat str stringexp expr notemptyprog prog
%token TRUE FALSE STRING VARIABLE
%token OPENPAREN CLOSEPAREN ACTSTR
%token DOTT HINT OPENBLOCK CLOSEBLOCK
%token SEMICOLON TRUE FALSE
%nonassoc EQ NE LT GT LE GE REGEXP
%left OR
%left AND
%right NOT
%nonassoc DEREF
%start grammarswitch
%%

grammarswitch: ACTSTR { keynote_exceptionflag = keynote_donteval = 0; } program
              {  /* XXX need to do something?
                printf("yacc:     RULE grammarswitch: ACTSTR program\n"); */
              }
```

```
program: prog {   //print_tree($1);
                  extract_DNF_tree($1);
                  //print_tree($1);
                  print_tree_file($1);
                  free_tree($1);
                }

prog:    /* Nada */ { $$ = NNULL; }
    | notemptyprog SEMICOLON prog
            {
              $$ = $1;
            }

notemptyprog: expr HINT afterhint
            {
              $$ = $1;
            }
    |  expr
            {
              $$ = $1;
            }

afterhint: str
            {
              /* XXX need to do something?
              printf("yacc: RULE afterhint: str\n");  */
            }
    | OPENBLOCK prog CLOSEBLOCK
            {
              /* XXX need to do something?
              $$ = $2; */
            }


expr:      OPENPAREN expr CLOSEPAREN
            {
              $$ = $2;
            }
    | expr AND expr
            {
              $$ = add_node("&&", $1, $3);
            }
    | expr OR expr
            {
              $$ = add_node("||", $1, $3);
            }
    | NOT expr
            {
              $$ = add_node("!", $2, NNULL);
            }
    | stringexp
            {
              $$ = $1;
            }
    | TRUE
            {
              $$ = add_leaf("STRING", "TRUE");
            }
    | FALSE
            {
              $$ = add_leaf("STRING", "FALSE");
            }
```

```
stringexp: str EQ str
           {
             //printf("yacc:stringexp: str EQ str\tadd EQ, str1: %s, str2: %s\n",
             //        $1->left.value, $3->left.value);
             $$ = add_node("==", $1, $3);
           }
    | str NE str
           {
             //printf("yacc: stringexp: str NE str\tadd NE, str1: %s, str2: %s\n",
             //        $1->left.value, $3->left.value);
             $$ = add_node("!=", $1, $3);
           }
    | str LT str
           {
             //printf("yacc: stringexp: str LT str\tadd LT, str1: %s, str2: %s\n",
             //        $1->left.value, $3->left.value);
             $$ = add_node("<", $1, $3);
           }
    | str GT str
           {
             //printf("yacc: stringexp: str GT str\tadd GT, str1: %s, str2: %s\n",
             //        $1->left.value, $3->left.value);
             $$ = add_node(">", $1, $3);
           }
    | str LE str
           {
             //printf("yacc: stringexp: str LE str\tadd LE, str1: %s, str2: %s\n",
             //        $1->left.value, $3->left.value);
             $$ = add_node("<=", $1, $3);
           }
    | str GE str
           {
             //printf("yacc: stringexp: str GE str\tadd GE, str1: %s, str2: %s\n",
             //        $1->left.value, $3->left.value);
             $$ = add_node(">=", $1, $3);
           }
    | str REGEXP str
           {
             /* XXX - unhandled case
             printf("yacc: RULE stringexp: str REGEXP str - unhandled case\n");
             ??? add a REGEXP node ??? */
           }

/* XXX - unhandled case, ignoring rule alltogether
str: str DOTT str     {
                   printf("yacc: RULE str: str DOTT str - unhandled case\n");
                   strcpy($$, $1);
                   strcpy($$ + strlen($1), $3);
                   free($1);
                   free($3); end of comment here
                     }
*/

str: strnotconcat
           {
             $$ = $1;
             /* XXX left default handling from keynote.y
                add a "string??" node */
           }
strnotconcat: STRING
           {
             $$ = add_leaf("STRING", $1);
           }
    | OPENPAREN str CLOSEPAREN
           {
             $$ = $2;
           }
    | VARIABLE
           {
             $$ = add_leaf("VARIABLE", $1);
           }
    | DEREF str
           {
             /* XXX - unhandled case
               printf("yacc: DEREF str - unhandled case\n");
               $$ = calloc(strlen('$') + strlen($2),
                                 sizeof(char));
               strcpy($$,'$');
               strcpy($$ + strlen('$'), $2);
             */
           }
%%
void
kderror(char *s)
{}

/*******************************************************************************
* nalloc
* Allocates memory for a node
* Output:
*     struct node* - pointer to the node
********************************************************************************/
struct node*
nalloc()
{
  struct node *np;

  /*XXX add: error checking for memory allocation np==-1*/
  np = (struct node *) malloc(sizeof(struct node));
  if (np == NNULL)
  {
    printf("nalloc: Out of Memory\n");
    return ((struct node*) -1);
  }
  return np;
}

/*******************************************************************************
* add_leaf
* Generates a leaf (a node with only left value filled)
* Inputs:
*     char *type -  type of info in leaf, can be STRING or VARIABLE
*     char *value -  actual value of STRING or VARIABLE
* Output:
*     struct node* - pointer to the leaf
********************************************************************************/
struct node*
add_leaf(char* type, char* value)
{
  /*XXX add: error checking for memory allocation np==-1*/
  struct node *np = nalloc();
  np->operator = type;
  np->left.value = value;
  //printf("adding leaf: %s\twith leaf value : %s\n", np->operator, np->left.value);
  return np;
}
```

```
/******************************************************************************
* add_node
* Generates a node
* Inputs:
*     char *op -  type of operator (can be &&,||,!,==,!=,>,>=,<,<=)
*     char node *left - pointer to the left child of node
*     char node *right - pointer to the right child of node
* Output:
*     struct node* - pointer to the node
*******************************************************************************/
struct node*
add_node(char* op, struct node *left, struct node *right)
{
  /*XXX add: error checking for memory allocation np==-1*/
  struct node *np = nalloc();
  np->operator = op;
  //printf("adding node: %s\n", np->operator);
  np->left.np = left;
  np->right.np = right;
  return np;
}

/******************************************************************************
* apply_double_negation
* Converts expression of type NOT (NOT x) to x
* Inputs:
*     struct node *snode - pointer to the node to which double negation will be
*                          applied
*******************************************************************************/
void
apply_double_negation(struct node* snode)
{
  struct node *child, *grand_child;

  //printf("\napply_double_negation");
  child = LCHILD(snode);
  grand_child = LCHILD(child);

  bcopy(grand_child, snode, sizeof(struct node));
  free(child);
  free(grand_child);
}

/******************************************************************************
* apply_and_DeMorgan
* Converts expression NOT (x AND y) to (NOT x) OR (NOT y)
* Inputs:
*     struct node *snode - root node that contains the NOT
*******************************************************************************/
void
apply_and_DeMorgan(struct node* snode)
{
  struct node *child, *grand_child1, *grand_child2, *new_child;

  child = LCHILD(snode);
  grand_child1 = LCHILD(child);
  grand_child2 = RCHILD(child);

  snode->operator = "||";
  child->operator = "!";
  RCHILD(child) = NNULL;
  new_child = add_node("!", grand_child2, NNULL);
  RCHILD(snode) = new_child;
}

/******************************************************************************
* apply_or_DeMorgan
* Converts expression NOT (x OR y) to (NOT x) AND (NOT y)
* Inputs:
*     struct node *snode - root node that contains the NOT
******************************************************************************/
void
apply_or_DeMorgan(struct node* snode)
/*XXX e apply_and_DeMorgan, apply_or_DeMorgan can be unified to one function through
snode->operator */
{
        struct node *child, *grand_child1, *grand_child2, *new_child;

        child = LCHILD(snode);
        grand_child1 = LCHILD(child);
        grand_child2 = RCHILD(child);

        snode->operator = "&&";
        child->operator = "!";
        RCHILD(child) = NNULL;
        new_child = add_node("!", grand_child2, NNULL);
        RCHILD(snode) = new_child;
}

/******************************************************************************
* inverse_tokens
* Converts expression NOT (x RELATIONAL_OPERATOR y)  to
*                       x INVERSE_RELATIONAL_OPERATOR y
* Inputs:
*     struct node *snode - root node that contains the NOT
******************************************************************************/
void
inverse_tokens(struct node* snode)
{
  struct node *child = NNULL;
  int relational = 0;

  child = LCHILD(snode);
  if (strcmp(child->operator, "==") == 0)
  {
    child->operator = "!=";
    relational = 1;
  }
  else if (strcmp(child->operator, "!=") == 0)
  {
    child->operator = "==";
    relational = 1;
  }
  else if (strcmp(child->operator, "<") == 0)
  {
    child->operator = ">=";
    relational = 1;
  }
  else if (strcmp(child->operator, ">") == 0)
  {
    child->operator = "<=";
    relational = 1;
  }
  else if (strcmp(child->operator, "<=") == 0)
  {
    child->operator = ">";
    relational = 1;
  }
  else if (strcmp(child->operator, ">=") == 0)
  {
    child->operator = "<";
    relational = 1;
  }
  if (relational)
  {
    bcopy(child, snode, sizeof(struct node));
    free(child);
  }
}

/******************************************************************************
* permeate_nots
* Converts expressions of type NOT (x AND y) to (NOT x) OR (NOT y)
*                              NOT (x OR y)     (NOT x) AND (NOT y)
* for the whole tree
* Inputs:
*     struct node *snode - root node of the tree in which NOTs
*                          will be permeated
******************************************************************************/
void
permeate_nots(struct node* snode)
{
  if (snode == NNULL)
  {
    return;
  }
  //if root node operator is a NOT
  if (strcmp(snode->operator, "!") == 0)
  {
    //printf("\npermeate_nots: found a NOT");
    if (LCHILD(snode) != NNULL)
    {
      //if left child operator is a NOT
      if (strcmp(LCHILD(snode)->operator, "!") == 0)
      {
        //remove the two NOTs and permeate
        apply_double_negation(snode);
        permeate_nots(snode);
      }
      else if (strcmp(LCHILD(snode)->operator, "&&") == 0)
        apply_and_DeMorgan(snode);
      else if (strcmp(LCHILD(snode)->operator, "||") == 0)
        apply_or_DeMorgan(snode);
      else
        //the node's operator is relational
        inverse_tokens(snode);
    }
  }
/*  if (strcmp(snode->operator, "VARIABLE") == 0)
  {
    printf("\npermeate_nots: found a VARIABLE");
    printf("\n\t snode->left.value %s", snode->left.value);
  }
  if (strcmp(snode->operator, "STRING") == 0)
  {
    printf("\npermeate_nots: found a STRING");
    printf("\n\t snode->left.value %s", snode->left.value);
  } */
  //if node's children are not leaves, permeate nots for them
  if ( (strcmp(snode->operator, "VARIABLE") != 0) &&
       (strcmp(snode->operator, "STRING") != 0) )
  {
    permeate_nots(LCHILD(snode));
    permeate_nots(RCHILD(snode));
  }
}

/******************************************************************************
* copy_bool_tree
* generates a copy of a tree
* Inputs:
*     struct node *snode - root node of tree to be copied
******************************************************************************/
struct node*
copy_bool_tree(struct node* snode)
{
  struct node *ctree;

  if (snode == NNULL)
  {
    return NNULL;
  }
  //copy root node
  /*XXX add: error checking for memory allocation ctree==-1*/
  ctree = nalloc();
  bcopy(snode, ctree, sizeof(struct node));
  //copy left child
  if ( (strcmp(LCHILD(snode)->operator, "VARIABLE") != 0) &&
       (strcmp(LCHILD(snode)->operator, "STRING") != 0) )
    if (LCHILD(snode) != NNULL)
      LCHILD(ctree) = copy_bool_tree(LCHILD(snode));
  //copy right child
  if ( (strcmp(RCHILD(snode)->operator, "VARIABLE") != 0) &&
       (strcmp(RCHILD(snode)->operator, "STRING") != 0) )
    if (RCHILD(snode) != NNULL)
      RCHILD(ctree) = copy_bool_tree(RCHILD(snode));
  return ctree;
}

/******************************************************************************
* distribute_branches
* Converts an expression of type (x OR y) AND z
*                                (x AND z) OR (y AND z)
* Inputs:
*     struct node *snode - root node of tree whose branches we distribute
******************************************************************************/
void
distribute_branches(struct node* snode)
{
  struct node *rchild, *cp_rchild, *gchild1, *gchild2;

  rchild = RCHILD(snode);
  cp_rchild = copy_bool_tree(rchild);

  gchild1 = LCHILD(LCHILD(snode));
  gchild2 = RCHILD(LCHILD(snode));

  snode->operator = "||";
  LCHILD(snode)->operator = "&&";
  RCHILD(snode) = add_node("&&", gchild2, rchild);
  RCHILD(LCHILD(snode)) = cp_rchild;
```

```
}

/*******************************************************************************
* and_distribute
* Converts recursively expressions of type (x OR y) AND z                   to
*                                          (x AND z) OR (y AND z)
* Inputs:
*     struct node *snode - root of tree on which distributive law will be
*                          applied
* Output:
*     int - 1 if distributive law was applied or if there was a change to
*           some part of tree, 0 otherwise
*******************************************************************************/
int
and_distribute(struct node* snode)
{
  int left, right;

  if (snode == NNULL)
  {
    return 0;
  }
  //if node is not an AND
  if (strcmp(snode->operator, "&&") != 0)
  {
    //if node is already an OR apply distributive law
    //to its children
    if (strcmp(snode->operator, "||") == 0)
    {
      left = and_distribute(LCHILD(snode));
      right = and_distribute(RCHILD(snode));
      return (left || right);
    }
    else
      return 0;
  }
  //if node is an AND
  else
  {
    int swap = 0;
    struct node *temp;
    //if right child's operator is an OR, swap positions of left and
    //right child (so if expression looked like      x AND (y OR z)
    //it becomes (y OR z) AND x
    /*XXX missing check that LCHILD isn't already an OR */
    if (strcmp(RCHILD(snode)->operator, "||") == 0)
    {
      /*XXX add: error checking for memory allocation temp==-1 */
      temp = nalloc();
      bcopy(RCHILD(snode), temp, sizeof(struct node));
      bcopy(LCHILD(snode), RCHILD(snode), sizeof(struct node));
      bcopy(temp, LCHILD(snode), sizeof(struct node));
      swap = 1;
      free(temp);
    }
    //if left child is an OR, convert (y OR z) AND x    to
    //                                (y AND x) OR (z AND x)
    if ( swap || (strcmp(LCHILD(snode)->operator, "||") == 0) )
    {
      distribute_branches(snode);

      and_distribute(snode);
      return 1;
    }
    //if left child is not an OR
    else
    {
      left = right = 0;
      //if left child is not a leaf, apply distributive law to it
      if ( (strcmp(LCHILD(snode)->operator, "VARIABLE") != 0) &&
           (strcmp(LCHILD(snode)->operator, "STRING") != 0) )
        left = and_distribute(LCHILD(snode));
      //if right child is not a leaf, apply distributive law to it
      if ( (strcmp(RCHILD(snode)->operator, "VARIABLE") != 0) &&
           (strcmp(RCHILD(snode)->operator, "STRING") != 0) )
        right = and_distribute(RCHILD(snode));
      return (left || right);
    }
  }
}

/*******************************************************************************
* distribute_DNF
* Applies distributive law to tree
* Inputs:
*     struct node* root -  root of boolean tree
*******************************************************************************/
void
distribute_DNF(struct node* snode)
{
  int distribution = 1;

  while (distribution)
  {
    distribution = and_distribute(snode);
  }
}

/*******************************************************************************
* extract_DNF_tree
* Converts tree to DNF form
* Inputs:
*     struct node* root -  root node of boolean tree to be converted to DNF
*******************************************************************************/
void
extract_DNF_tree(struct node* root)
{
  permeate_nots(root);
  distribute_DNF(root);
}

/*******************************************************************************
* binary_node
* Checks if node has a logical or relational operator involving two operands
* Inputs:
*     struct node* snode - node to be checked
* Output:
*     int - 1 if node has two operands, 0 otherwise
*******************************************************************************/
int
binary_node(struct node* snode)
{
  if ((strcmp(snode->operator, "&&") == 0) || (strcmp(snode->operator, "||") == 0) ||
      (strcmp(snode->operator, "==") == 0) || (strcmp(snode->operator, "!=") == 0) ||
      (strcmp(snode->operator, "<") == 0) || (strcmp(snode->operator, ">") == 0) ||
      (strcmp(snode->operator, "<=") == 0) || (strcmp(snode->operator, ">=") == 0))
    return 1;
  else
    return 0;
}

/******************************************************************************
* print_binary_node
* Prints a binary node to screen: left child first, operator, right child last
* Inputs:
*     struct node* snode - binary node to be printed
*******************************************************************************/
void
print_binary_node(struct node* snode)
{
  //print left child
  if (snode->left.np != NNULL)
    print_tree(snode->left.np);
  //print operator
  printf(" %s ", snode->operator);
  //print right child
  if (snode->right.np != NNULL)
    print_tree(snode->right.np);
}

/******************************************************************************
* print_unary_node
* Prints a unary node to screen: operator first, left child last
* Inputs:
*     struct node* snode - unary node to be printed
*******************************************************************************/
void
print_unary_node(struct node* snode)
{
  //print operator
  printf("%s", snode->operator);
  //print left child
  if (snode->left.np != NNULL)
    print_tree(snode->left.np);
}

/******************************************************************************
* print_tree
* Prints tree to screen
* Inputs:
*     struct node* snode - root of tree to be printed
*******************************************************************************/
void
print_tree(struct node* snode)
{
  if (snode == NNULL)
    return;
  //if node is leaf print it
  if (strcmp(snode->operator, "STRING") == 0)
    printf("\"%s\"",snode->left.value);
  else if (strcmp(snode->operator, "VARIABLE") == 0)
    printf("%s",snode->left.value);
  //if node with two children
  else if (binary_node(snode))
  {
    printf("(");
    print_binary_node(snode);
    printf(")");
  }
  //if node with one child
  else if (strcmp(snode->operator, "!") == 0)
  {
    printf("(");
    print_unary_node(snode);
    printf(")");
  }
}

/*******************************************************************************
* fprint_binary_node
* Prints a binary node to a file: left child first, operator, right child last
* Inputs:
*     FILE* file - file to output node to
*     struct node* snode - binary node to be printed
********************************************************************************/
void
fprint_binary_node(FILE* file, struct node* snode)
{
  //print left child
  if (snode->left.np != NNULL)
    fprint_tree(file, snode->left.np);
  //print operator
  fprintf(file, " %s ", snode->operator);
  //print right child
  if (snode->right.np != NNULL)
    fprint_tree(file, snode->right.np);
}

/*******************************************************************************
* fprint_unary_node
* Prints a unary node to a file: operator first, left child last
* Inputs:
*     FILE* file - file to output node to
*     struct node* snode - unary node to be printed
********************************************************************************/
void
fprint_unary_node(FILE* file, struct node* snode)
{
  //print operator
  fprintf(file, "%s", snode->operator);
  //print left child
  if (snode->left.np != NNULL)
    fprint_tree(file, snode->left.np);
}

/*******************************************************************************
* fprint_tree
* Prints tree to a file
* Inputs:
*     FILE* file - file to output tree to
*     struct node* snode - root of tree to be printed
********************************************************************************/
void
fprint_tree(FILE* file, struct node* snode)
{
  if (snode == NNULL)
    return;
  //if node is leaf print it
  if (strcmp(snode->operator, "STRING") == 0)
    fprintf(file, "\"%s\"",snode->left.value);
  else if (strcmp(snode->operator, "VARIABLE") == 0)
    fprintf(file, "%s",snode->left.value);
  //if node with two children
  else if (binary_node(snode))
  {
    fprintf(file, "(");
    fprint_binary_node(file, snode);
    fprintf(file, ")");
  }
  //if node with one child
  else if (strcmp(snode->operator, "!") == 0)
  {
    fprintf(file, "(");
    fprint_unary_node(file, snode);
    fprintf(file, ")");
  }
}

/*******************************************************************************
* print_tree_file
* Prints.tree to the file described by the global variable targetdnf
* Inputs:
*     struct node* snode - root of tree to be printed
********************************************************************************/
void
print_tree_file(struct node* snode)
{
  FILE *file;

  if ((file=fopen(targetdnf, "w")) != NULL)
  {
    fprint_tree(file, snode);
    fclose(file);
  }
  else
    /*XXX Incomplete error handling*/
    printf("\nprint_tree_file: error in opening file %i", errno);
}

/*******************************************************************************
* free_tree
* Frees memory aloocated for tree
* Inputs:
*     struct node* root - root node of tree to be freed
********************************************************************************/
void
free_tree(struct node* root)
{
  if (root == NNULL)
  {
    return;
  }
  //recursively free children
  //if node has two children, free both of them
  if (binary_node(root))
  {
    free_tree(root->left.np);
    free_tree(root->right.np);
  }
  //if node has only one child, free this left child only
  else if (strcmp(root->operator, "!") == 0)
    free_tree(root->left.np);
  /*freeing a leaf or the root*/
  /*if ((strcmp(root->operator, "STRING") == 0) || (strcmp(root->operator, "VARIABLE")
== 0))
```

```
    printf("\nfreeing a leaf : %s", root->left.value);
  else
    printf("\nfreeing a root node: %s", root->operator);*/
  free(root);
  return;
}
```

APPENDIX C

Header File: `tree.h`

```
#ifndef __TREE_H__
#define __TREE_H__

/***************************************************************************
* node
* There can be two types of nodes: intermediate and leaf/terminal nodes
* An intermediate node uses:
*    operator,      which can be:  a logical or relational operator
*    left,          which contains:
*                            np   pointer to left child node
*    and may also use (if the operator is binary)
*    right,         which contains:
*                            np   pointer to right child node
* (Fields not mentioned are empty/ignored)
* A leaf node has:
*    operator,      which can be:  a VARIABLE or STRING indicator
*    left,          which contains:
*                            value   actual value of VARIABLE or STRING
* (Fields not mentioned are empty/ignored)
****************************************************************************/
struct node
{
       char* operator;
       union
       {
              char* value;
              struct node *np;
       } left, right;
};

char* targetdnf;

#define LCHILD(snode)       ((snode)->left.np)
#define RCHILD(snode)       ((snode)->right.np)
#define NNULL       ((struct node *) 0)

#endif /* __TREE_H__ */
```

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
8725 John J. Kingman Rd., STE 0944
Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library, Code 013 2
Naval Postgraduate School
Monterey, CA 93943-5100

3. Research Office, Code 09 1
Naval Postgraduate School
Monterey, CA 93943-5138

4. Dr. Douglas Maughan 1
Defense Advanced Research Projects Agency
Advanced Technology Office
3701 North Fairfax Drive
Arlington, VA 22203-1714

5. Dr. Cynthia E. Irvine 10
Code CS/Ic
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943-5118

6. Evdoxia Spyropoulou 2
Eirinis 49-51
153 41, Agia Paraskevi
Athens
Greece

5. Timothy E. Levin 2
Code CS/Lt
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943-5118