CS 598D Formal Methods in Networking
Princeton University

Lecture 17-18:
Network Configuration Verification and Analysis Using BDDs

Ehab Al-Shaer

Cyber Defense and Network Assurability (CyberDNA) Center

School of Computing & Informatics

University of North Carolina, Charlotte, NC

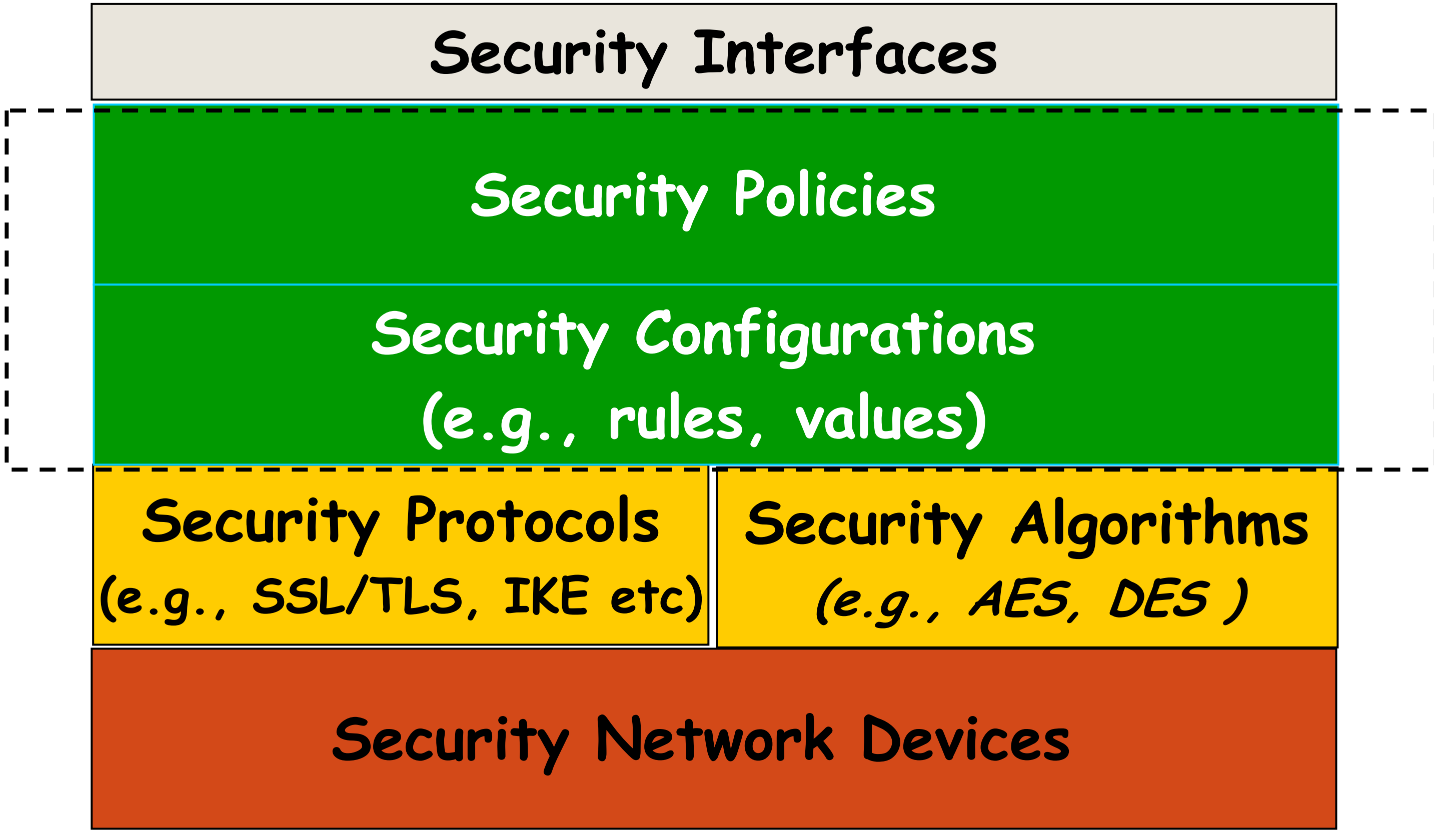
April 5,9, 2010

Acknowledgment: *Will Marrero, Hazem Hamed, and Adel El-Atawy*

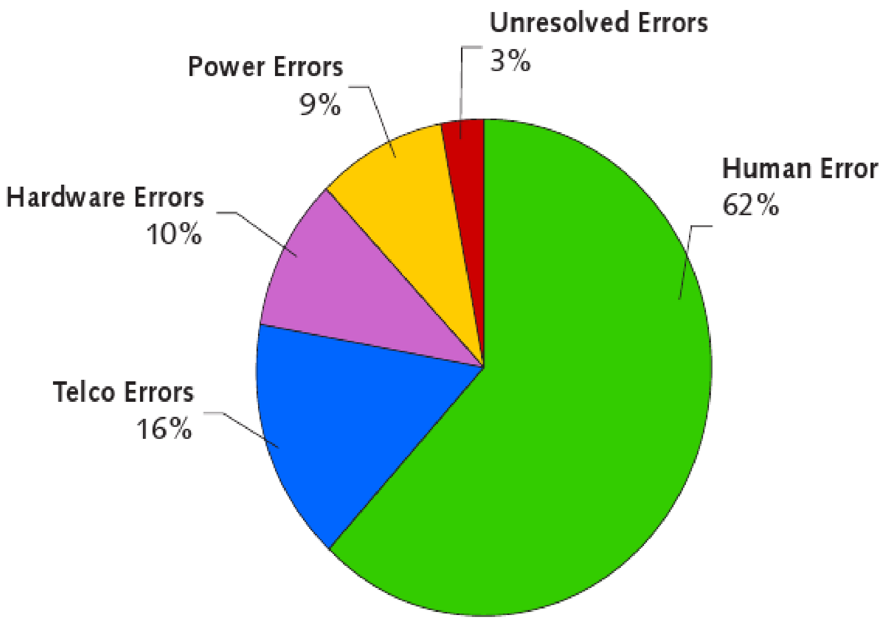
Lecture 17 & 18 Outline

- Lecture 17
  - Network Configuration Challenges
  - The need of abstraction in network configuration
  - Limitation of set-theoretic approach
  - Introduction to BDD Configuration Conflict Analysis (ConfigLego)
  - Policy Hardening and Optimization
- Lecture 18
  - ConfigChecker: Global End-to-End Network Security Configuration Verification
  - Examples
  - Future research agenda

Role of Security Polices & Configurations



State of Network Configuration Management



"Eighty percent of IT budgets is used to maintain the status quo.", Kerravala, Zeus. "As the Value of Enterprise Networks Escalates, So Does the Need for Configuration Management." *The Yankee Group* January 2004 [2].

"Most of network outages are caused by operators errors rather than equipment failure.", Z. Kerravala. Configuration Management Delivers Business Resiliency. The Yankee Group, November 2002.

- "It is estimated that configuration errors enable 65% of cyber attacks and cause 62% of infrastructure downtime", Network World, July 2006.
- *Recent surveys show Configuration errors are a large portion of operator errors which are in turn the largest contributor to failures and repair time [1].*
- *"Management of **ACLs** was the most critical missing or limited feature, Arbor Networks' Worldwide Infrastructure Security Report, Sept 2007.*

[1] D. Oppenheimer, A. Ganapathi, and D. A. Patterson. Why Internet services fail and what can be done about these? In *USENIX USITS*, Oct. 2003.

Challenges of Network Security Configuration

- **Security Systems are composed of: Algorithms + Protocols + Configuration**
- **Network security devices are policy-based (ACL) devices**
  - A policy P is a set of Rules, s.t. R:<proto><srcIP><srcP><destIP><destP> … ➔ <action>
- **Scale challenge due to large number of devices and rules**
  - Policies might have *large number of inter-related* rules in a single device (15K rules)
  - Policies are *distributed, yet inter-connected* forming a global security policy
  - Heterogeneous (multi-vendor) security devices
- **Operational semantic Challenge due to different device roles**
  - Rule-order semantics vs. recursive ACL
  - Single-trigger vs. multi-trigger policies
  - Binary vs. multi-value action
- **Network dynamic challenge due to failures or traffic engineering**
  - Multi-domain administration ➔ conflicts due to uncoordinated policy changes

Intra-Firewall Conflicts

- Shadowing

$$R_x[\text{order}] < R_y[\text{order}], R_x \Re_{\text{EM}} R_y, R_x[\text{action}] \neq R_y[\text{action}]$$

$$R_x[\text{order}] < R_y[\text{order}], R_x \Re_{\text{IM}} R_y, R_x[\text{action}] \neq R_y[\text{action}]$$

- Correlation

$$R_x \Re_{\text{C}} R_y, R_x[\text{action}] \neq R_y[\text{action}]$$

- Exception

$$R_x[\text{order}] < R_y[\text{order}], R_y \Re_{\text{IM}} R_x, R_x[\text{action}] \neq R_y[\text{action}]$$

- Redundancy

$$R_x[\text{order}] < R_y[\text{order}], R_x \Re_{\text{EM}} R_y, R_x[\text{action}] = R_y[\text{action}]$$

$$R_x[\text{order}] < R_y[\text{order}], R_y \Re_{\text{IM}} R_x, R_x[\text{action}] = R_y[\text{action}]$$

$$R_x[\text{order}] < R_y[\text{order}], R_x \Re_{\text{IM}} R_y, R_x[\text{action}] = R_y[\text{action}] \text{ and}$$
$$\nexists R_z \text{ where } R_x \Re_{\{\text{IM,RC}\}} R_z, R_x[\text{order}] < R_z[\text{order}], R_x[\text{action}] \neq R_z[\text{action}]$$

- Irrelevance

The path from R_x[src] to R_x[dst] is not controlled by the firewall

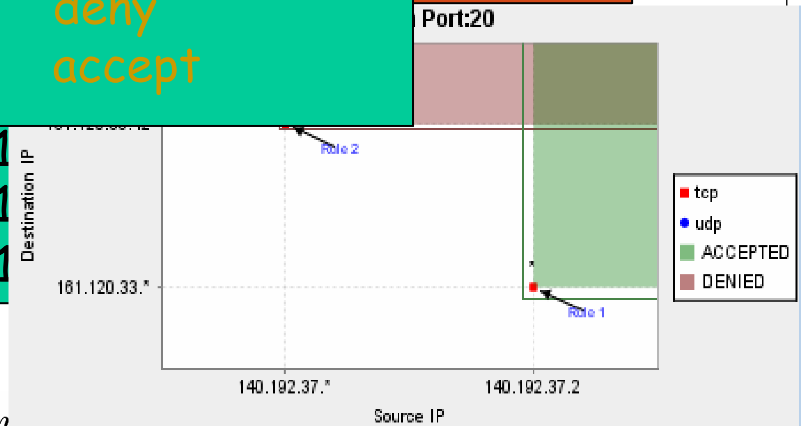
Intra-Firewall Conflicts

R1: Allow CS to access Registration server
R2: Block Students from accessing Administration-Domain
R1 < R2: Students are **ALLOWED** to access the Registration server
R1 ... : Students are **BLOCKED** to access the Registration server
Si... CS-faculty accessing the Financial-server?

y: udp, 140.192.33.*, any, 161.121.27.*, 53 deny
x: udp, 140.192.33.40, any, 161.121.27.40, 53 accept

- Excep...

x: udp, 140.192.33.40, any, 161.121
Z: udp, 140.192.33.40, any, 161.121
y: udp, 140.192.*.*, any, 161.121



- Redundancy

$$R_x[\text{o}\ldots] < R_y[\text{order}], R_x \Re_{\text{EM}} R_y, \ldots [\text{action}] = \ldots [\text{action}]$$

$$R_x[\text{order}] < R_y[\text{order}], R_y \Re_{\text{IM}} R_x, R_x[\text{action}] = R_y[\text{action}]$$

$$R_x[\text{order}] < R_y[\text{order}], R_x \Re_{\text{IM}} R_y, R_x[\text{action}] = R_y[\text{action}] \text{ and}$$

$$\nexists R_z \text{ where } R_x \Re_{\{\text{IM,RC}\}} R_z, R_x[\text{order}] < R_z[\text{order}], R_x[\text{action}] \neq R_z[\text{action}]$$

- Irrelevance

The path from R_x[src] to R_x[dst] is not controlled by the firewall

Formalization of Inter-Firewall Conflicts

- Shadowing

$R_d \Re_{\mathsf{EM}} R_u,\ R_u[\text{action}]=\text{deny},\ R_d[\text{action}]=\text{accept}$

$R_d \Re_{\mathsf{IM}} R_u,\ R_u[\text{action}]=\text{deny},\ R_d[\text{action}]=\text{accept}$

$R_u \Re_{\mathsf{IM}} R_d,\ R_u[\text{action}]=\text{deny},\ R_d[\text{action}]=\text{accept}$

$R_u \Re_{\mathsf{IM}} R_d,\ R_u[\text{action}]=\text{accept},\ R_d[\text{action}]=\text{accept}$

- Spuriousness

$R_u \Re_{\mathsf{EM}} R_d,\ R_u[\text{action}]=\text{accept},\ R_d[\text{action}]=\text{deny}$

$R_u \Re_{\mathsf{IM}} R_d,\ R_u[\text{action}]=\text{accept},\ R_d[\text{action}]=\text{deny}$

$R_d \Re_{\mathsf{IM}} R_u,\ R_u[\text{action}]=\text{accept},\ R_d[\text{action}]=\text{deny}$

$R_d \Re_{\mathsf{IM}} R_u,\ R_u[\text{action}]=\text{accept},\ R_d[\text{action}]=\text{accept}$

$R_u \Re_{\mathsf{IM}} R_d,\ R_u[\text{action}]=\text{deny},\ R_d[\text{action}]=\text{deny}$

- Redundancy

$R_d \Re_{\mathsf{EM}} R_u,\ R_u[\text{action}]=\text{deny},\ R_d[\text{action}]=\text{deny}$

$R_d \Re_{\mathsf{IM}} R_u,\ R_u[\text{action}]=\text{deny},\ R_d[\text{action}]=\text{deny}$

- Correlation

$R_u \Re_{\mathsf{C}} R_d,\ R_u[\text{action}]=\text{accept},\ R_d[\text{action}]=\text{accept}$

$R_u \Re_{\mathsf{C}} R_d,\ R_u[\text{action}]=\text{deny},\ R_d[\text{action}]=\text{deny}$

$R_u \Re_{\mathsf{C}} R_d,\ R_u[\text{action}]=\text{accept},\ R_d[\text{action}]=\text{deny}$

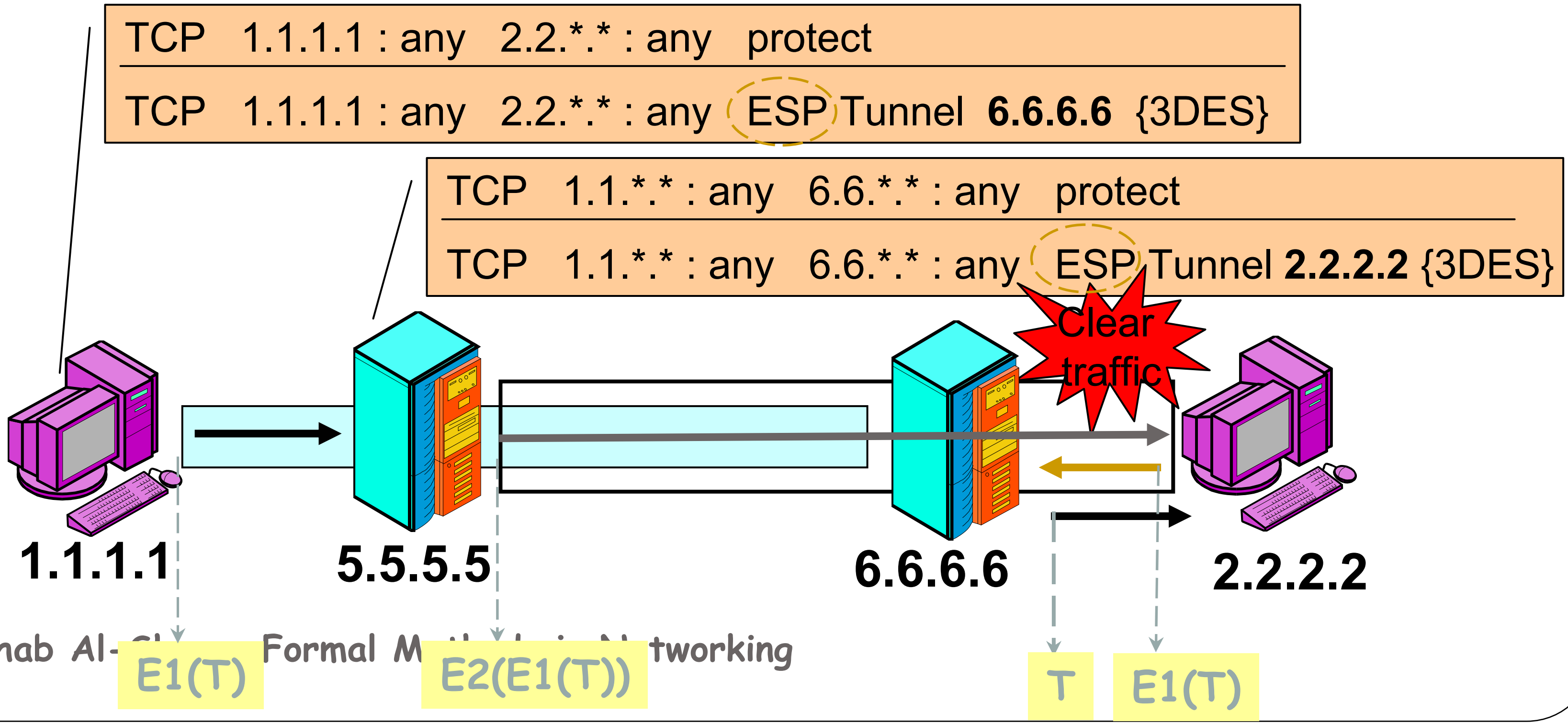
$R_u \Re_{\mathsf{C}} R_d,\ R_u[\text{action}]=\text{deny},\ R_d[\text{action}]=\text{accept}$

Uses binary actions & Pair-wise analysis → Does Not Scale

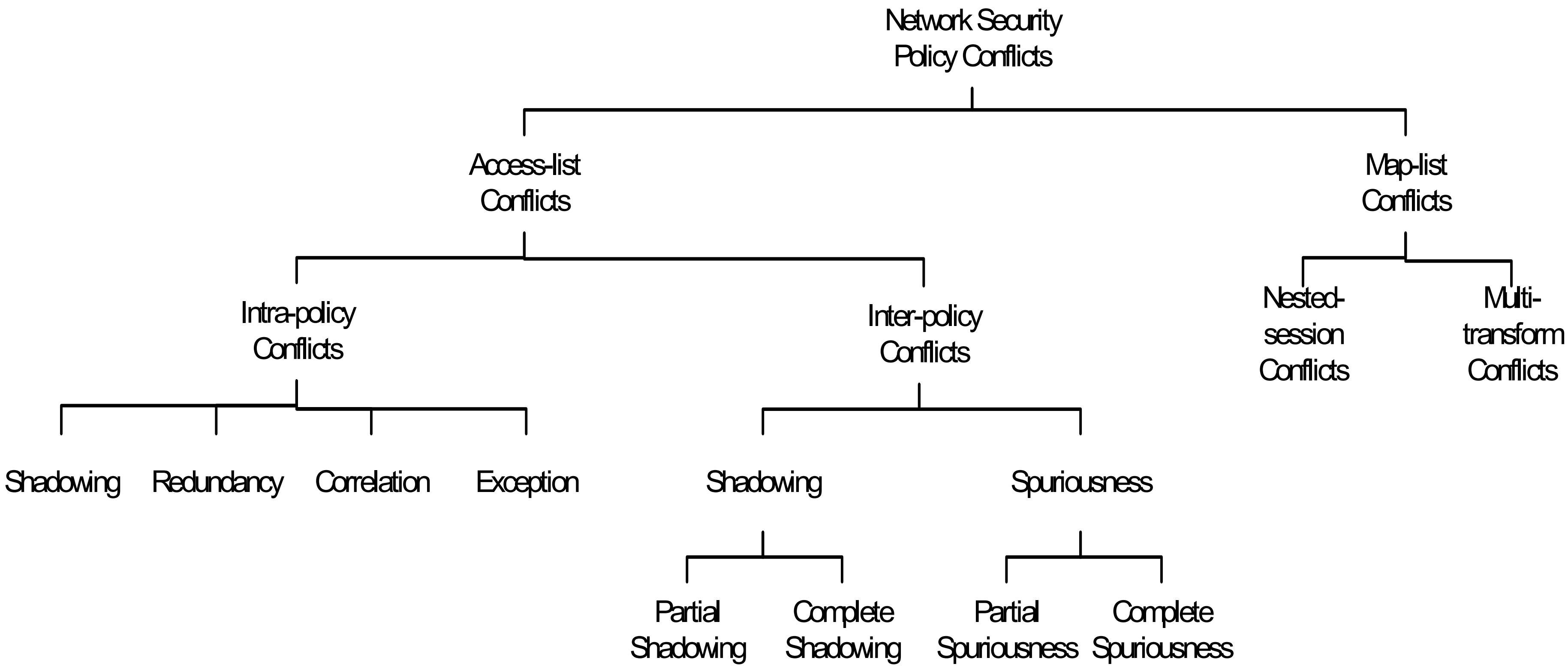
IPSec Inter-Policy Overlapped-tunnel Misconfiguration

- Overlapping tunnels with shared/common traffic
- Traffic decapsulated in reverse order to traffic flow

$$R_i^u[src\_ip] \subseteq R_j^d[src\_ip] \text{ and } R_i^u[tunnel\_dst] \subseteq R_j^d[dst\_ip] \text{ and}$$
$$\text{Location}(R_i^u[tunnel\_dst]) < \text{Location}(R_j^d[tunnel\_dst])$$



Taxonomy of Conflicts in Firewall and IPSec Policies

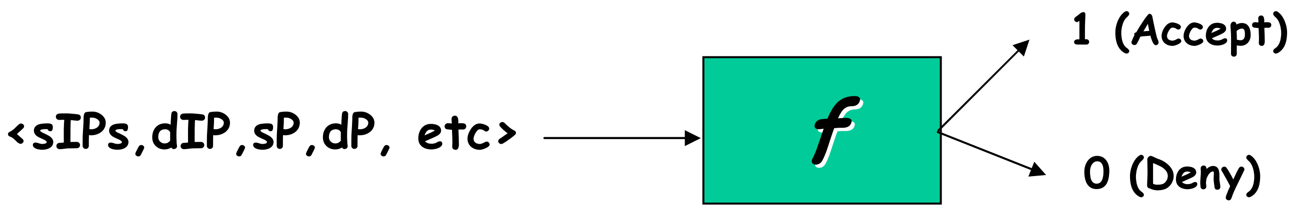


***IEEE Communication Magazine, Ehab Al-Shaer and Hazem Hamed, April 2006**

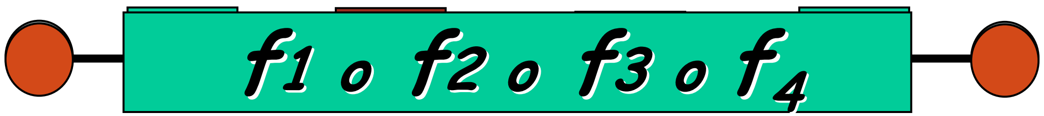
Looking for a better abstraction

- Limitation of the Set-Theoretic approach
  - Multi-actions will cause exponential growth in conditions.
  - It requires pair-wise analysis of rules
  - It can not be generalized to other ACL devices such as IPSec where multi-trigger and recursive actions are uses
  - It does not support abstraction and composability
- Objectives:
  - Unified/canonical abstraction for different policy semantic
  - Composability
  - Property-based verification
  - Scalability

Modeling Access Control Configuration as Boolean Formulas



- Evaluate
- Compare
- Compose



Modeling ACL Configuration Using BDDs

- An ACL policy is a sequence of filtering rules that determine the appropriate action to take for any incoming packets: $P = R1, R2, R3, .., Rn$
- Each rule can be written in the form:

$$R_i := C_i \rightsquigarrow a_i$$

where C_i is the constraint on the filtering fields that must be satisfied in order to trigger the action a_i

- The condition C_i can be represented as a Boolean expression of the filtering fields $f_1, f_2, \ldots, f_k$ as follows:

$$C_i = fv_1 \wedge fv_2 \wedge \cdots \wedge fv_k$$

where each fv_j expresses a set of matching field values for field f_j in rule R_i. Thus, we can formally describe a ACL policy as:

$$P_a = (C_1 \wedge b_1) \vee (\neg C_1 \wedge C_2 \wedge b_2) \ldots \vee (\neg C_1 \wedge \neg C_2 \ldots \neg C_{i-1} \wedge C_i \wedge b_i)$$

rule1 rule2 rule$_n$

$$\text{where } b_i = \begin{cases} 1 \text{ if } action_i = a \\ 0 \text{ if } action_i \neq a \end{cases}$$

Concise Formalization

- Single-trigger policy **is an access policy where only one action is triggered for a given packet.** C_i is the 1st match leads to action *a*

$$P_a = \bigvee_{i \in index(a)} (\neg C_1 \wedge \neg C_2 \ldots \neg C_{i-1} \wedge C_i)$$

$$P_a = \bigvee_{i \in index(a)} \bigwedge_{j=1}^{i-1} \neg C_j \wedge C_i$$

- Multiple-trigger policy **is an access policy where multiple different actions may be triggered for the same packet.** C_i is any match leads to action *a*

$$P_a = \bigvee_{i \in index(a)} C_i$$

where $$index(a) = \{i \mid R_i = C_i \rightsquigarrow a\}$$

Introduction to BDD

Boolean variables and functions:

- A boolean variable x is a variable ranging over the range 0 and 1.

- A boolean function f of n arguments is a function from $\{0,1\}^n$ to $\{0,1\}$, $f(n) : B^n \rightarrow B$.

- There are many ways to represent a boolean function.

Boolean functions representation:

- A boolean function *f* can be represented by:
  - Truth tables.
  - Propositional formulas.
  - Disjunctive Normal Form (DNF), in which a formula is a disjunctions of conjunctions of literals.
  - Conjunctive Normal Form (CNF), in which a formula is a conjunctions of disjunctions of literals.
  - Binary Decision Diagrams (BDDs) (If-Else Normal Form or INM)
    - $x \rightarrow y1, y2 \Leftrightarrow (x \wedge y1) \vee (\neg x \wedge y2)$
    - E.g., $\neg x$ is $(x \rightarrow 0,1)$

What is a BDD?

- BDD is a simpler form of Binary decision trees where:
  - Non-terminal nodes are labeled with boolean variables *x, y, z* …
  - Terminal nodes are labeled with either 0 or 1.
  - Each non-terminal node has two edges, one dashed line and one solid line.
  - Dashed line from node *x* is called low(*x*) while the solid line is called high(*x*).

- ***Reduced (O)BDD*** *iff*
  - ***Uniqueness:*** *if var(u)=var(v) , low(u)=low(v), high(u)=high(v)* → u=v
  - ***Non-redundant test:*** *low(u) = high(u) (v and u are different nodes)*

Boolean functions representation:

Representation of		test for		boolean operations		
boolean functions	compact?	satisf'ty	validity	$\cdot$	$+$	$\bar{}$
Prop. formulas	often	hard	hard	easy	easy	easy
Formulas in DNF	sometimes	easy	hard	hard	easy	hard
Formulas in CNF	sometimes	hard	easy	easy	hard	hard
Ordered truth tables	never	hard	hard	hard	hard	hard
Reduced OBDDs	often	easy	easy	medium	medium	easy

Figure: Boolean functions representations [CS]

Representing Boolean Functions

Formula:

$$(a \vee c) \wedge (b \rightarrow d)$$

Normal forms:

$$(a \vee c) \wedge (\neg b \vee d)$$

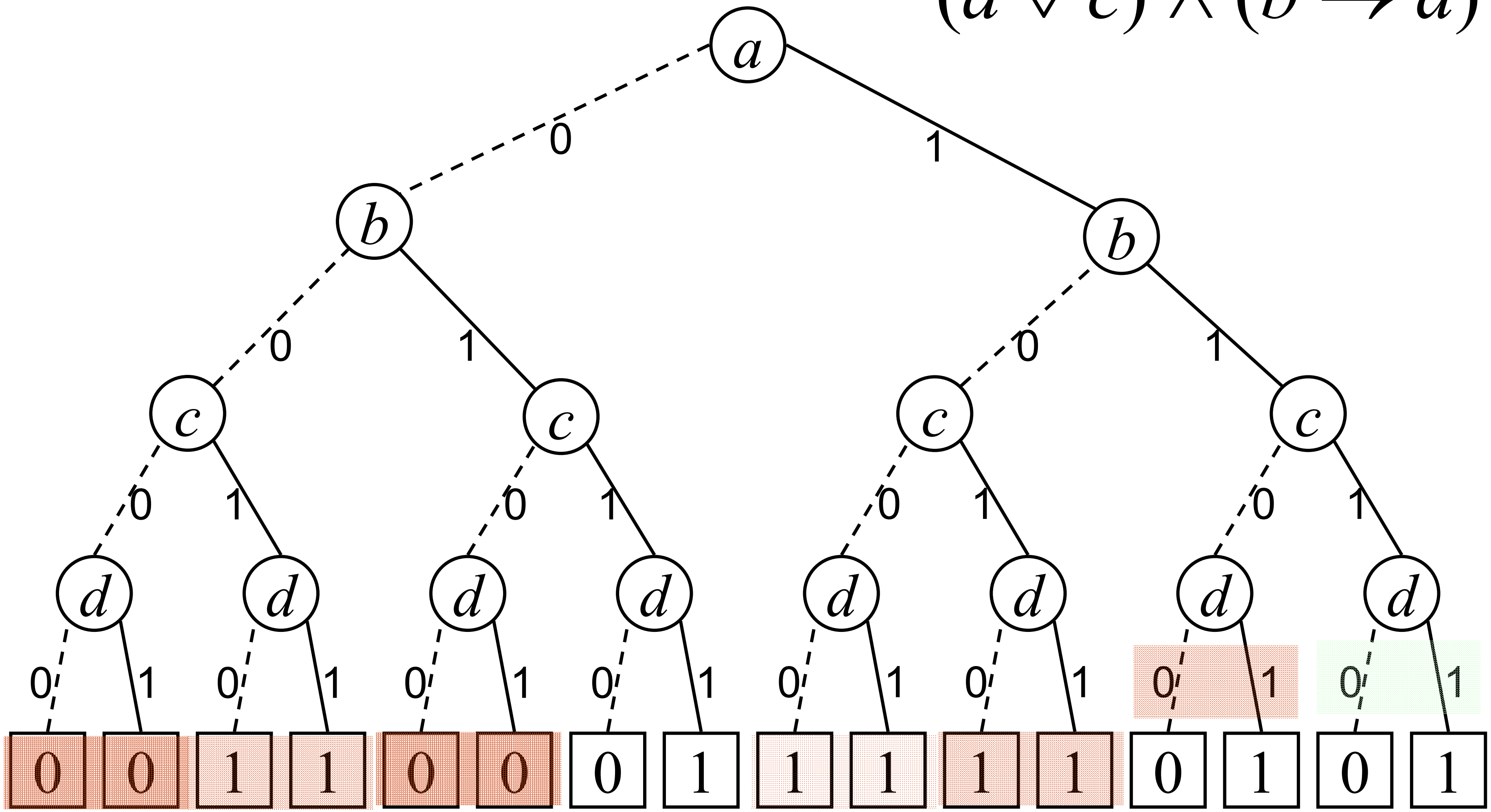
$$(a \wedge \neg b) \vee (a \wedge d) \vee (c \wedge \neg b) \vee (c \wedge d)$$

Truth table:

a	b	c	d	f
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

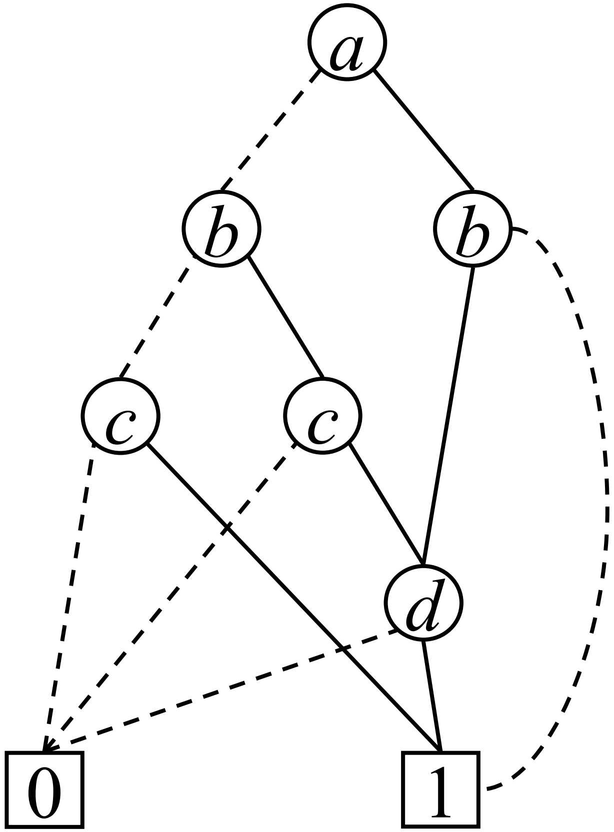
Binary Decision Tree

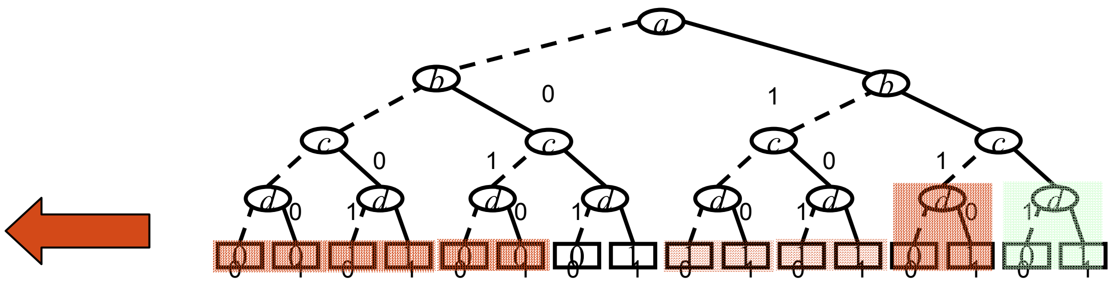
$$(a \vee c) \wedge (b \rightarrow d)$$



Ordered Binary Decision Diagram

$$(a \vee c) \wedge (b \rightarrow d)$$





BDD and truth tables

- The main disadvantage of truth tables is the space needed to maintain it.
- if we have 100 variables, we need 2^{100} entries in the table.
- in trees, we still need 2^n space to maintain it.
- why are BDDs useful?
    - some reductions can be done.

Reducing Decision Trees

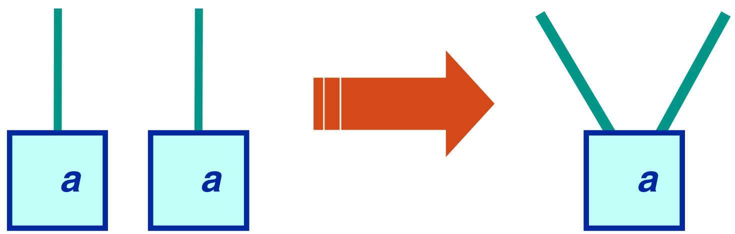
Two ways of simplifying decision trees:

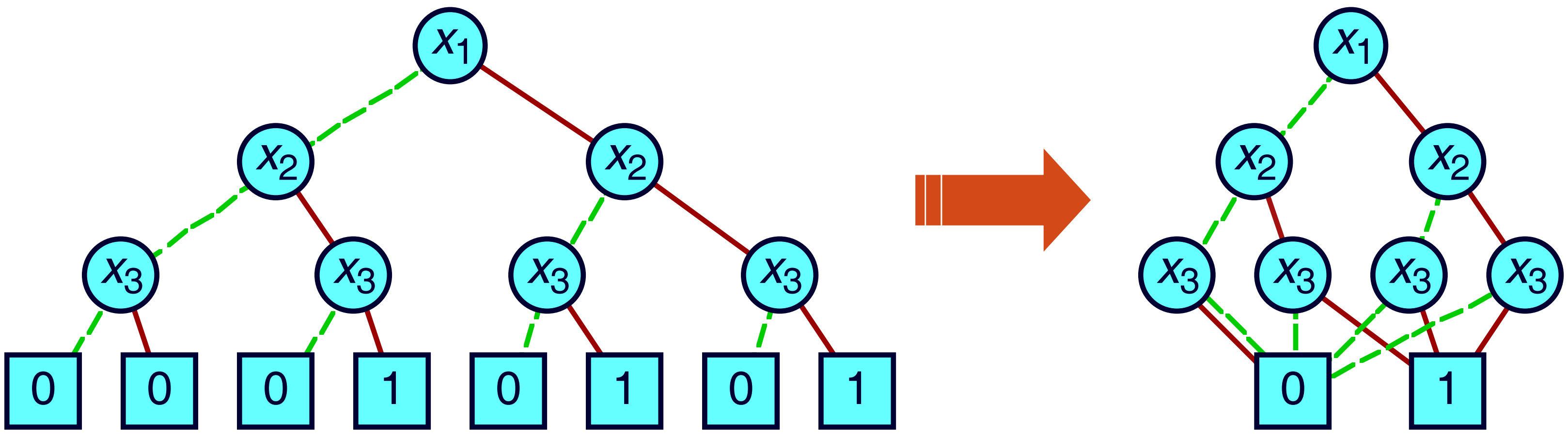
1. Identify and share identical subtrees.
2. Remove nodes whose left and right child nodes are identical.

Results in a Reduced Ordered Binary Decision Diagram (OBDD).

Reduction Rule #1:

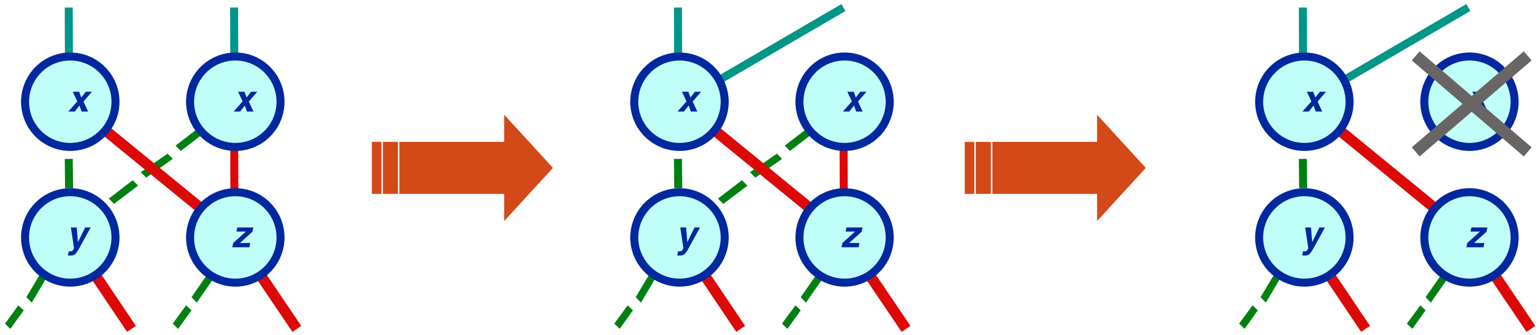
Merge equivalent leaves

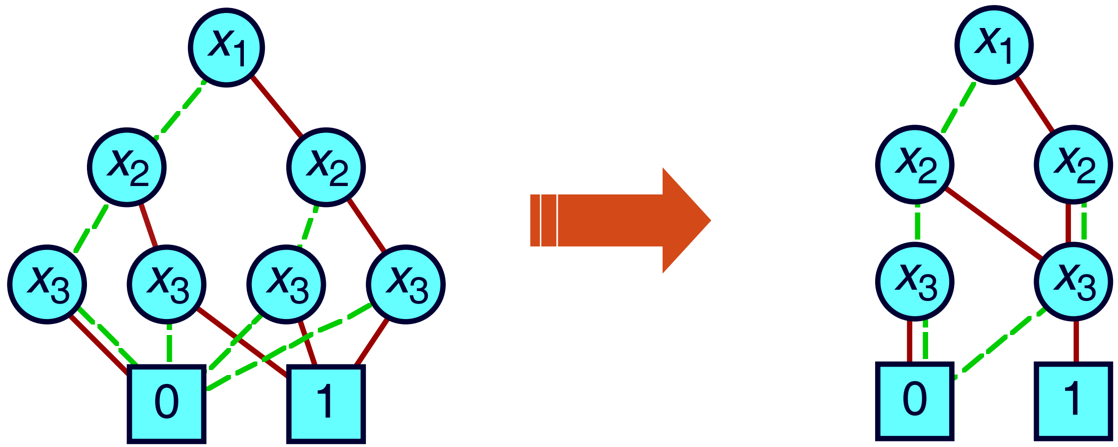




From [DP]

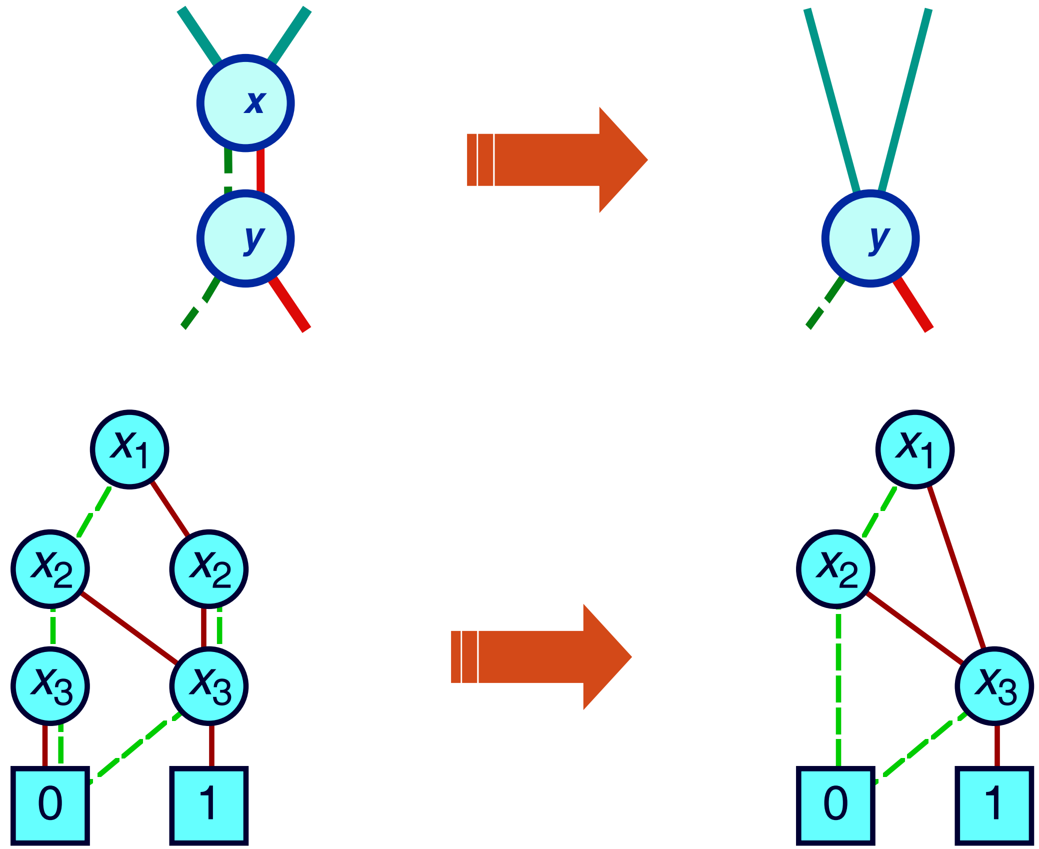
Reduction Rule #2: Merge isomorphic nodes





From [DP]

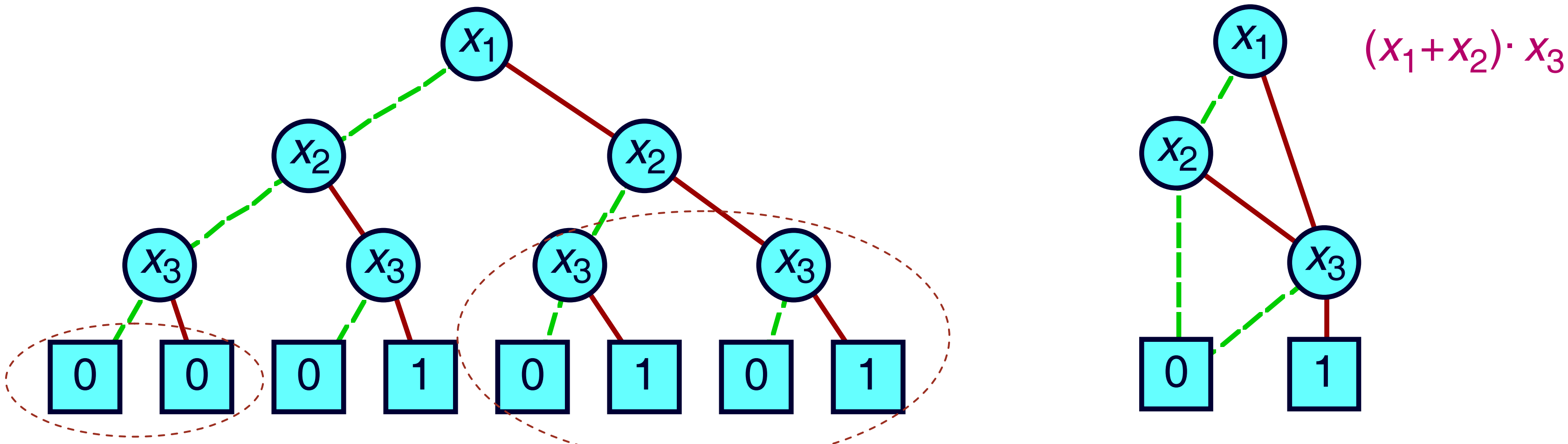
Reduction Rule #3: Eliminate Redundant Tests



From [DP]

Example OBDD

Initial Graph **Reduced Graph**



- Canonical representation of Boolean function
  - For given variable ordering
  - Two functions equivalent if and only if graphs isomorphic
    - Can be tested in linear time
  - Desirable property: *simplest form is canonical*.

From [DP]

Properties of BDD

Storage Efficiency (often compact)

Many common Boolean functions have small OBDD representations.

Canonicity

If the order in which the variables are tested is fixed, then there exists only one OBDD for each Boolean formula.

- **Lemma 1:** (Canonicity lemma)

For every function $f : Bn \rightarrow B$, there is **exactly one** ROBDD u with variable ordering x1<x2<…<xn such that $fu = f(x1, x2, \ldots, xn)$

Efficient operations

data structure for propositional logic formulas

- BDD operations: Build, Apply, Restrict, Existential quantification. SATCount, anySAT, allSAT

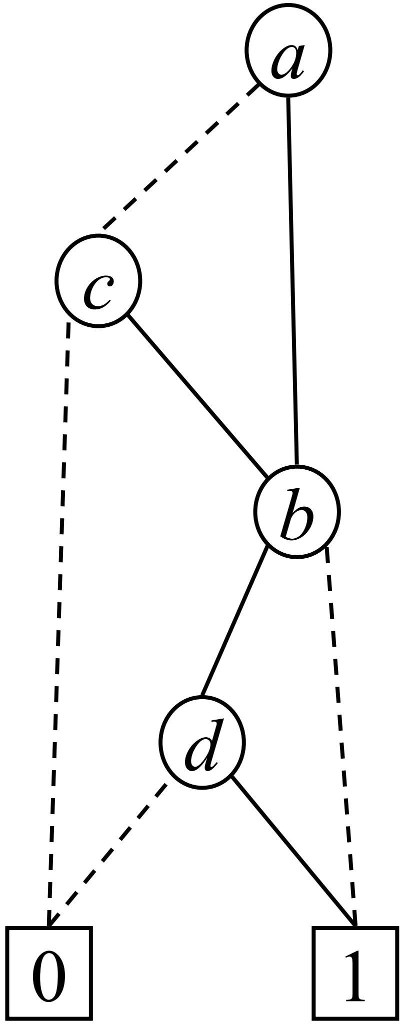
The Variable Ordering

On every branch in an OBDD, the variables must be tested in the same order, e.g.,

$$a < b < c < d$$

Different variable orderings yield different OBDDs.

Ordered Binary Decision Diagram



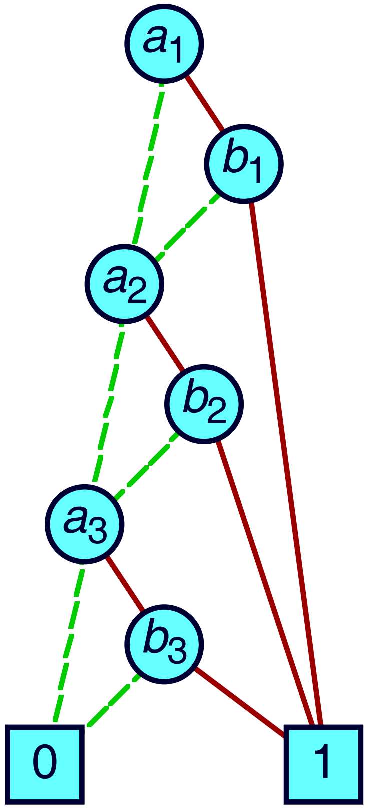
$$(a \vee c) \wedge (b \rightarrow d)$$

$$a < c < b < d$$

Effect of Variable Ordering

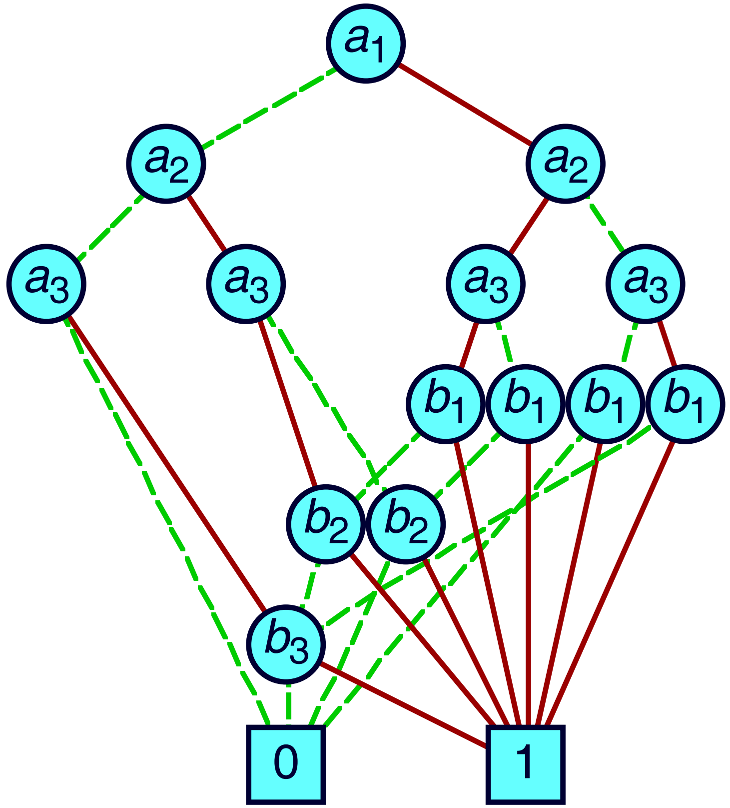
$$(a_1 \wedge b_1) \vee (a_2 \wedge b_2) \vee (a_3 \wedge b_3)$$

Good Ordering



Linear Growth

Bad Ordering



Exponential Growth

From [DP]

Now, APPLY (1/3)

- Let v_1, v_2 denote that root nodes of f_1, f_2, respectively, with $\text{var}(v_1) = x_1$ and $\text{var}(v_2) = x_2$.

1. If v_1 and v_2 are leafs, $f_1 \star f_2$ is a leaf node with value $\text{val}(v_1) \star \text{val}(v_2)$

$$0 \lor 1 = 1$$

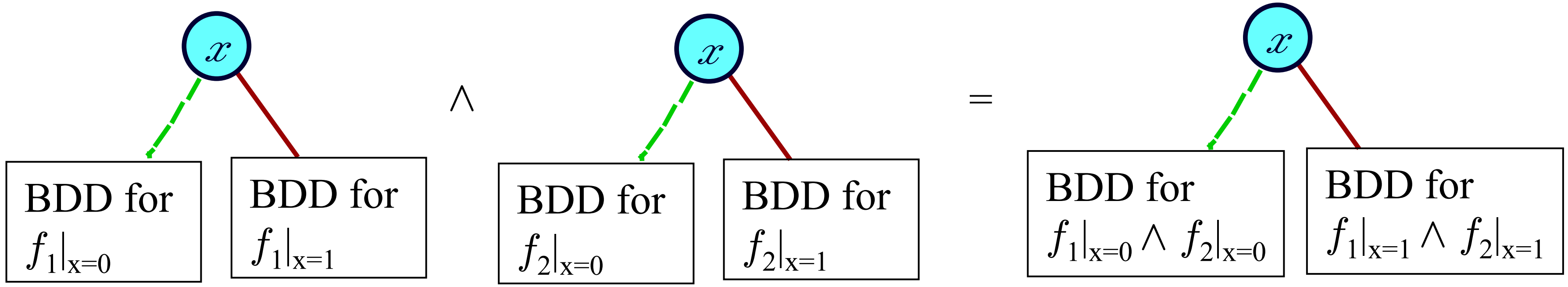
$$0 \land 1 = 0$$

From [DP]

Now, APPLY (2/3)

2. If $x_1 = x_2 = x$, apply Shanon expansion:

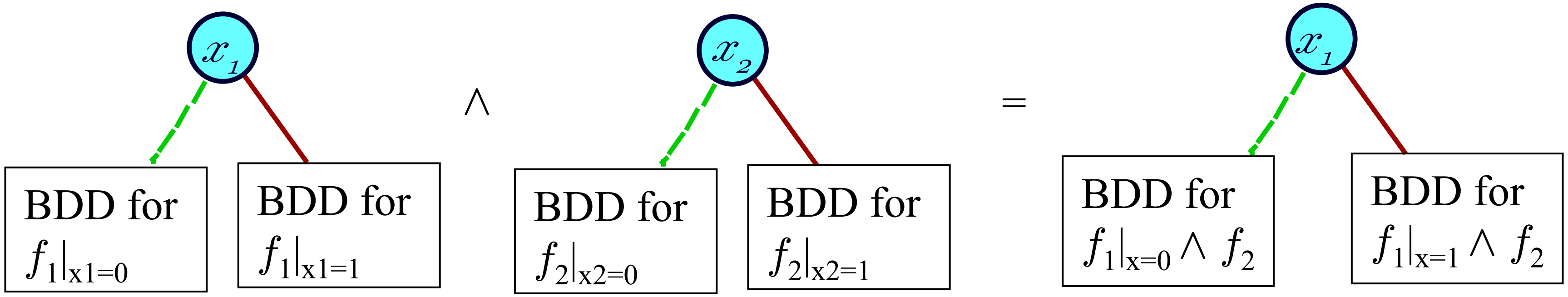
$$f_1 \star f_2 = (\neg x \wedge f_1|_{x=0} \star f_2|_{x=0} \vee x \wedge f_1|_{x=1} \star f_2|_{x=1})$$



From [DP]

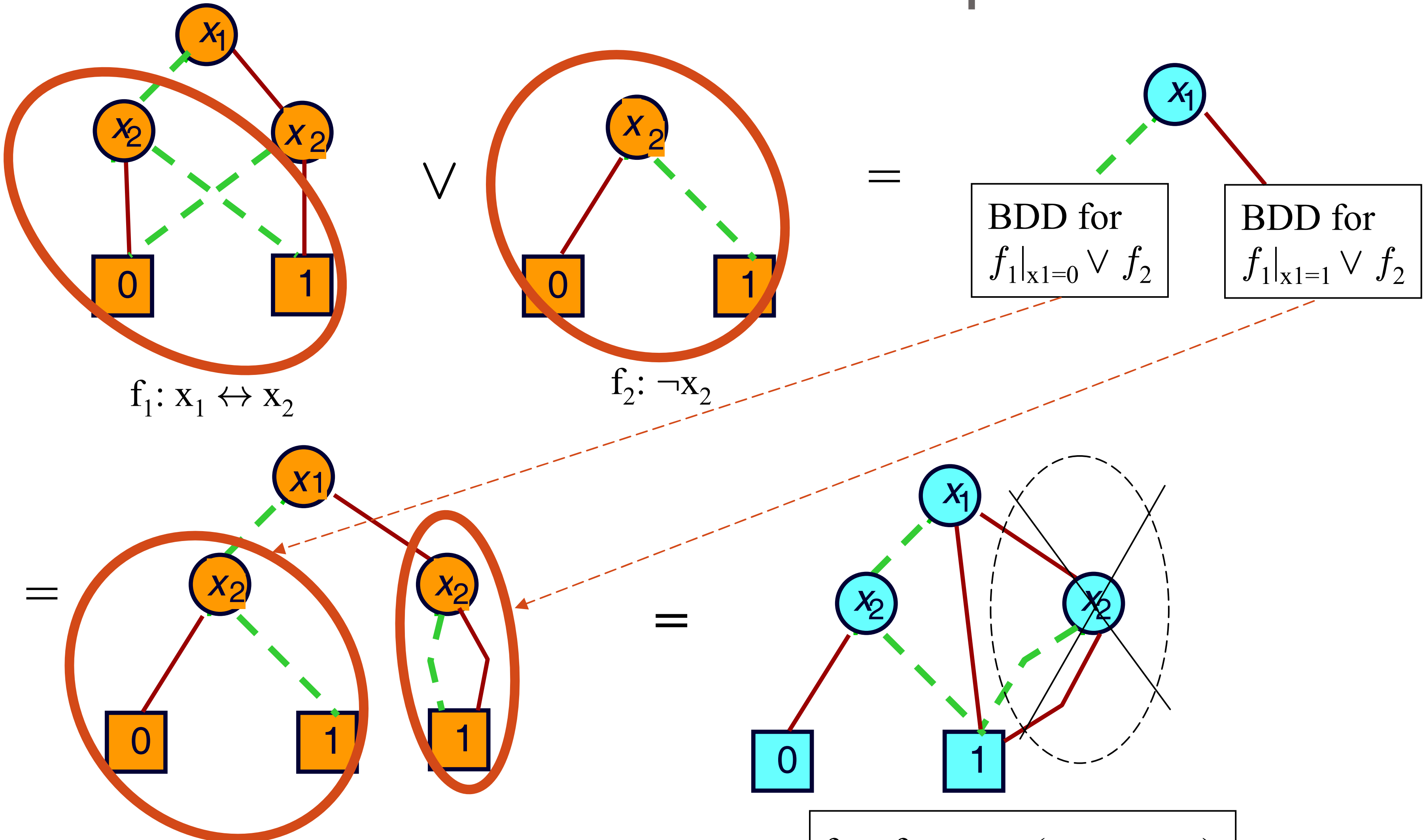
Now, APPLY (3/3)

3. else, suppose $x_1 < x_2$ in the variable order.
   $f_1 \star f_2 = (\neg x_1 \wedge f_1|_{x=0} \star f_2 \vee x_1 \wedge f_1|_{x=1} \star f_2)$



From [DP]

BDDs from below: example.



$f_1 \vee f_2 = x_1 \vee (\neg x_1 \wedge \neg x_2)$

BDD Operations

- Negation: $\neg B$
- Apply
  - OR $B_1 \bigvee B_2$
  - And $B_1 \bigwedge B_2$
  - Imply $B_1 \rightarrow B_2$
  - Equivalence $B_1 \rightarrow B_2$
- Restrict
  - Restrict(1, x, B) = f[1/x]
  - Restrict(0, x, B) = f[0/x]
- Existential quantifier

$$\exists x f = f[0/x] \vee f[1/x] = apply(+, rstrcit(0, x, B_f), restrict(1, x, B_f))$$

- Universal quantifier

$$\forall x f = f[0/x] \wedge f[1/x] = apply(., rstrcit(0, x, B_f), restrict(1, x, B_f))$$

Hints about Variable Ordering

- May not impact the BDD for some (few) problems
  - E.g., parity check
- But it often matters (see previous examples)
- Finding the optimal variable ordering for minimum BDD size is computationally hard (NP complete)
- Many good heuristic obtains often work (built-in in Buddy)
  - Keep correlated variable close
  - Use interleaving variable (x0y0x1y1 ..)
- Application-Based Heuristics
  - Exploit characteristics of application
  - e.g., Ordering for functions of combinational circuit
    - Traverse circuit graph depth-first from outputs to inputs

BDD operations running time

MK(i, u_0, u_1)	$O(1)$	
BUILD(t)	$O(2^n)$	
APPLY(op, u_1, u_2)	$O(\lvert u_1\rvert \lvert u_2\rvert)$	
RESTRICT(u, j, b)	$O(\lvert u\rvert)$	See note
SATCOUNT(u)	$O(\lvert u\rvert)$	See note
ANYSAT(u)	$O(\lvert p\rvert)$	$p = AnySat(u)$, $\lvert p\rvert = O(\lvert u\rvert)$
ALLSAT(u)	$O(\lvert r\rvert * n)$	$r = AllSat(u)$, $\lvert r\rvert = O(2^{\lvert u\rvert})$
SIMPLIFY(d, u)	$O(\lvert d\rvert \lvert u\rvert)$	See note

Note: These running times only holds if dynamic programming is used

Table 1: Worst-case running times for the ROBDD operations. The running times are the expected running times since they are all based on a hash-table with expected constant time search and insertion operations.

OBDD Packages

CUDD

`http://vlsi.colorado.edu/~fabio`

Buddy (what we used)

`http://buddy.sourceforge.net`

JDD (pure Java)

`http://javaddlib.sourceforge.net`

BDD Applications in Network Configuration Analysis

Applications

- **Conflict Detection**

(2) Configuration Hardening

Intra-Policy Conflicts Formalization : Crypto-access List

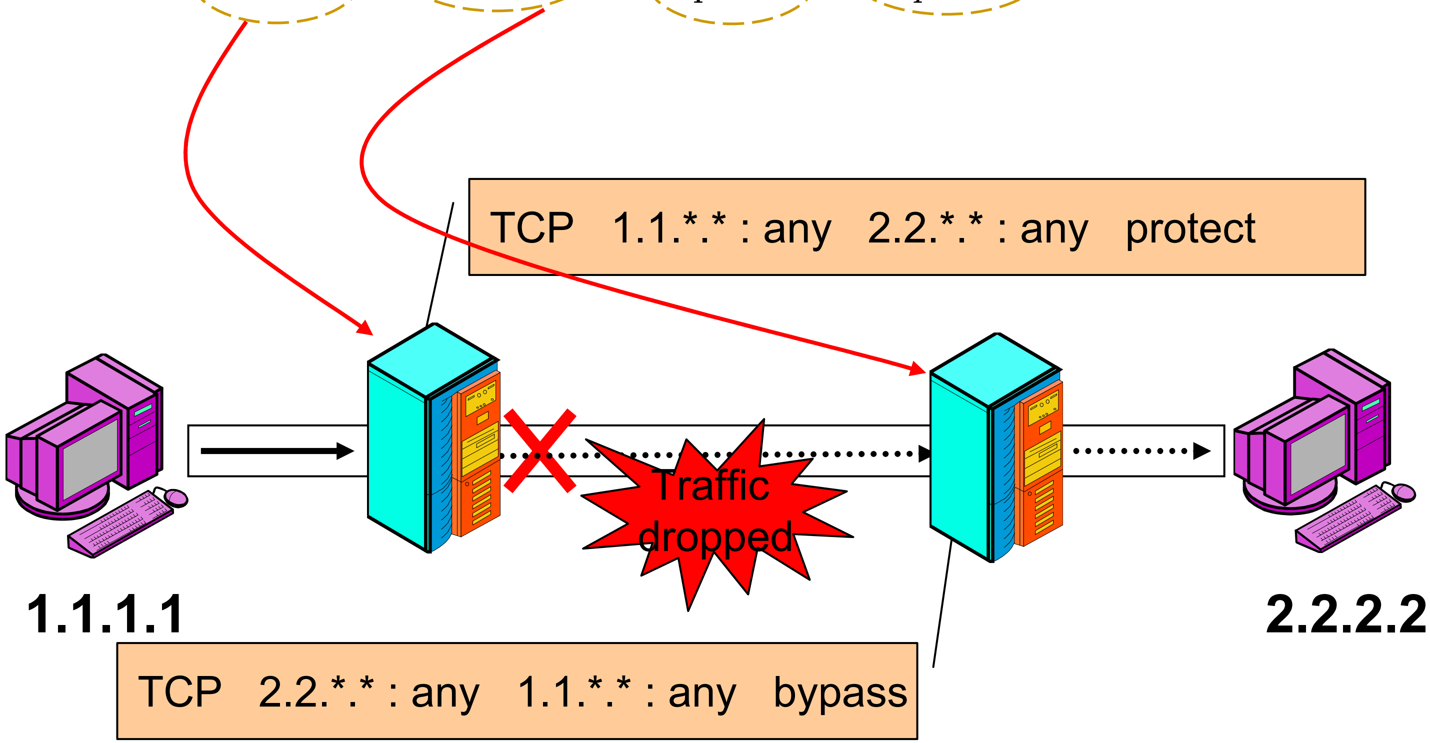
Soundness & Completeness

- **Policy expression S_a represents a policy that incorporates rule R_i, and S'_a is the policy with R_i excluded. R_i may be involved in the following conflicts:**
  - **Shadowing:**
    $[(S'_{a_i} \Leftrightarrow S_{a_i}) = true]$ and $[(C_i \Rightarrow S'_{a_i}) = false]$
  - **Redundancy:**
    $[(S'_{a_i} \Leftrightarrow S_{a_i}) = true]$ and $[(C_i \Rightarrow S'_{a_i}) \neq false]$
  - **Exception:**
    $[(S'_{a_i} \Leftrightarrow S_{a_i}) \neq true]$ and $[(C_i \Rightarrow S'_{a_i}) = false]$
  - **Correlation:**
    $[(S'_{a_i} \Leftrightarrow S_{a_i}) \neq true]$ and $[(C_i \Rightarrow S'_{a_i}) \neq false]$

IPSec Inter-Policy Conflicts Formalization: Crypto-access Lists

- **Shadowing**: upstream policy blocks traffic

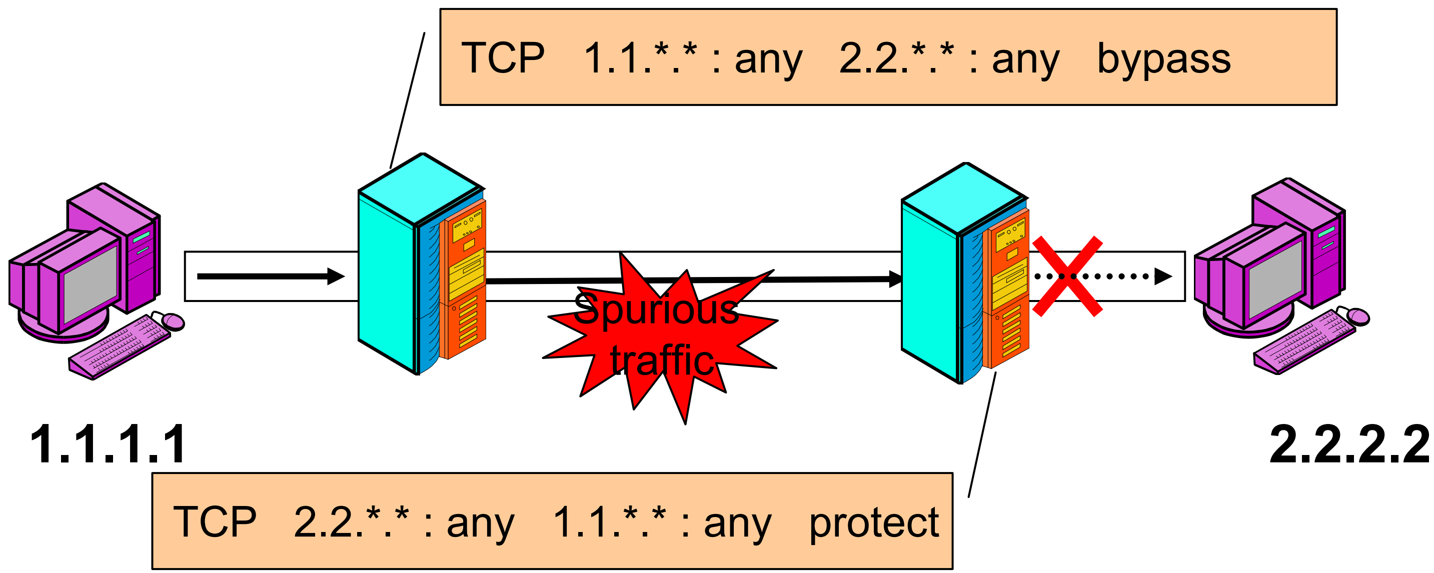
$$[(S^u_{discard} \wedge \neg S^d_{discard}) \vee (S^u_{protect} \wedge \neg S^d_{protect})] \neq false$$



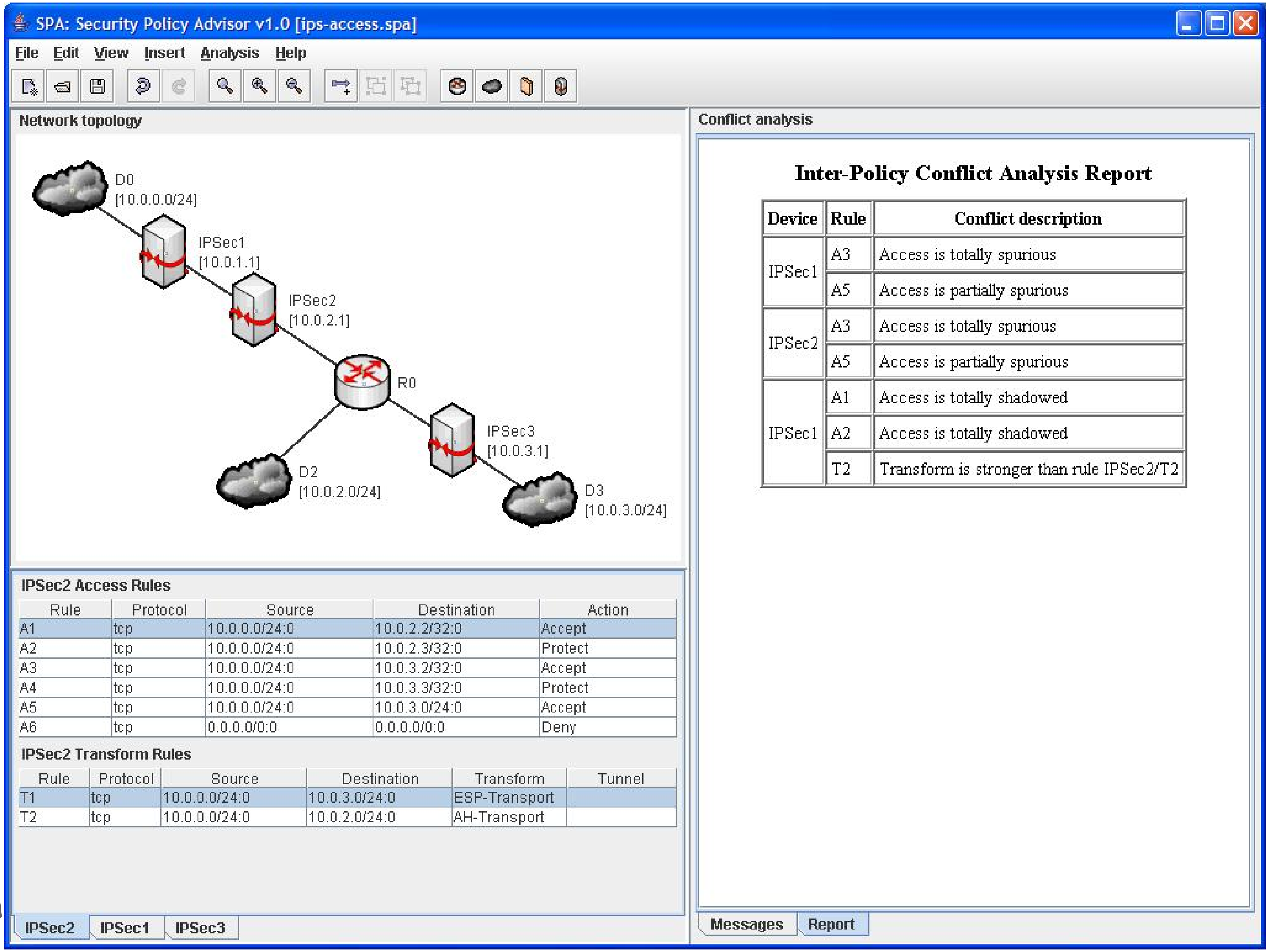
IPSec Inter-Policy Conflicts Formalization: Crypto-access Lists cont.

- **Spurious**: downstream policy blocks traffic

$$[(S^u_{bypass} \wedge \neg S^d_{bypass}) \vee (S^u_{protect} \wedge S^d_{discard})] \neq false$$



Security Policy Advisor Tool for Distributed Firewall & IPSec



Companies and Institutions Using Security Policy Advisor

- **Companies:**
  - Lisle Technology Partners, USA; Phontech, Norway; Naval Surface Warfare Center, Panama City, USA; Cisco Systems, USA; At&T, USA; Gateshead Council, UK; Danet Group, Germany; TNT Express Worldwide, UK Ltd, United Kingdom; Checkpoint, USA; FireWall-1, The Netherlands; DataConsult, Lebanon; Rosebank Consulting, GB; Mayer Consulting, USA; Panduit Corp, USA; UPMC Paris 5 University, France; Royal institute of Science, Sweden; GE, US; Aligo, USA; Motorola, Inc., USA; Landmark communications, inc., us; uekae.tubitak.gov, Turkey; Duke Energy, USA; The Midland Co, USA; NITW,INDIA; Deloitte & Touche LLP, US; National Taiwan University, Taiwan; Eircom.net. Irland; GE CF, USA; AIT, Thailand; Celestica, Thailand; and Others not listed
- **Universities/Institutions:**
  - ISRC, Queensland University of Technology, Australia; Imperial College and UCL, London, UK; Columbia University, USA; Georgia Institute of Technology ;NCSU, USA; USC, USA; University of Pittsburgh, PA; University of Waterloo, Canada; University Student in Cyprus International University, Cyprus; University of Rochester, US; UQAM, University of Quebec in Montreal, Canada; Saarland University, Germany; Technical University of Berlin, Computer Science Department, Germany; UCSB, US; Edith Cowan University, Australia; Universitat Oberta de Catalunya, Spain; ISG, Tunisia; York U, Toronto, Canada; Universidade Federal do Rio Grande do Sul, Brazil; UCL, Belgium; Kent State University, USA; UFRGS, Brazil; University of Stuttgart, IKR, Germany;

Composable Security Configuration Verification & Analysis

Themes:

- Security Configuration Hardening
- Integrating other device and host configuration
- Property based verification

Modeling Routing Access Control

- We can define the routing policies as follows: let a routing rule be encoded as $R_i := D_i \rightsquigarrow n$
  - Where *n* is integer representing the forwarding port ID

where D_i is the destination and n_i is a unique integer (id) designating the next hope in the network. Thus, the policy of the routing entries (ordered based on longest-common prefix) that forward to next hope n_k can be defined as follows:

$$T_n = \bigvee_{i \in index(n)} \bigwedge_{j=1}^{i-1} \neg D_j \wedge D_i \;\; s.t. \;\; index(n) = \{i \mid R_i = D_i \rightsquigarrow n\}$$

- We can then represent the entire routing table for a node *j* as follows: $T^j = \bigvee_{\forall n = next \;\; hope} T_n$

Composability: Path Conflict Analysis for Firewalls

- ***Lemma:*** **If S_A^u, S_A^d are the upstream and downstream firewalls in a path, then**
  (a) S^u causes inter-policy shadowing with S^d *iff* $[(\neg S_A^u \wedge S_A^d) \neq false]$
  (b) S^u causes inter-policy spuriousness with S^d *iff* $[(S_A^u \wedge \neg S_A^d) \neq false]$
- ***Lemma:*** **Shadow-free and spurious-free are *transitive* relations. Thus, assume S_A^i, S_A^j and S_A^k are upstream to downstream firewall polices in a path a, the following relation is always true (shadowing-free case) :**

$$[(\neg S_A^i \wedge S_A^j) = false] \bigwedge [(\neg S_A^j \wedge S_A^k) = false] \Rightarrow [(\neg S_A^i \wedge S_A^k) = false]$$

- Path Conflict: **Assuming S_A^1 to S_A^n are the firewall policies from upstream to downstream in the path from *x* to *y*, a *path conflict (x,y)* between any two firewalls from *i* to *n* path is defined as follows:**

(a) Path-Shadowing (x,y):

$$[\bigvee_{i=1,n-1 \ and \ i \in path(x,y)} \neg S_A^i \wedge S_A^{i+1} \neq false]$$

(b) Path-Spuriousness (x,y):

$$[\bigvee_{i=1,n-1 \ and \ i \in path(x,y)} S_A^i \wedge \neg S_A^{i+1} \neq false]$$

Diagnosing Unreachablility Problems between Routers and Firewalls

- Flow-level Analysis: **Is the flow C_k that is** forwarded **by routers in path *P* (each routing tables is represented as BDD T^i_j for router *i* and port j) but** blocked **due to conflict between *Routing* and *FW Filtering*:**

$$[(C_k \Rightarrow \bigwedge_{(i,j)\in P} T^i_j) \wedge (C_k \Rightarrow \neg S^n_A)] \neq false$$

  - This shows that a traffic C_j is forwarded by the routing policy, T^i_j, from node *i* to *n* but yet blocked by the filtering policy, $S^n_{discard}$, of the destination domain.
- Path-level Analysis: **What are all** unreachability Conflicts **between *Routing* and *Filtering*:**

$$\phi_k \leftarrow [SAT^*(\bigwedge_{(i,j)\in path(P)} T^i_j \wedge \neg S^n_A \wedge \neg(\bigwedge_{i=1,k-1} \phi_i))] \neq false$$

  - For phi=1, n misconfiguration examples, and phi(0) = ture
- Network or Federated-level Analysis: **Spurious conflict between downstream *d* and upstream *u* ISP domains:**

$$[(S^u_{bypass} \wedge \neg S^d_{bypass}) \vee (S^u_{limit} \wedge S^d_{discard})] \neq false$$

  - Notice that $S_{discard}$, S_{bypass} and S_{limit} are filtering policies representations related to the filtering actions as described in [POLICY08, ICNP05, CommMag06].

*: AnySAT

Automating Hardening of Security Configuration

Security Hardening & Intrusion Response

- Given the Boolean formula P that represents the configuration of the entire network, k, with variables $v_1, .. ,v_n$, what are all configuration changes to block *all* attack scenarios a_i without violating the requirements H_i
  - Example of A_i: (* -> telnetServer/23) and (ftpServer/any -> SQLServer/550)
  - Example of H_i: (SQLServer/* -> DNS/51)

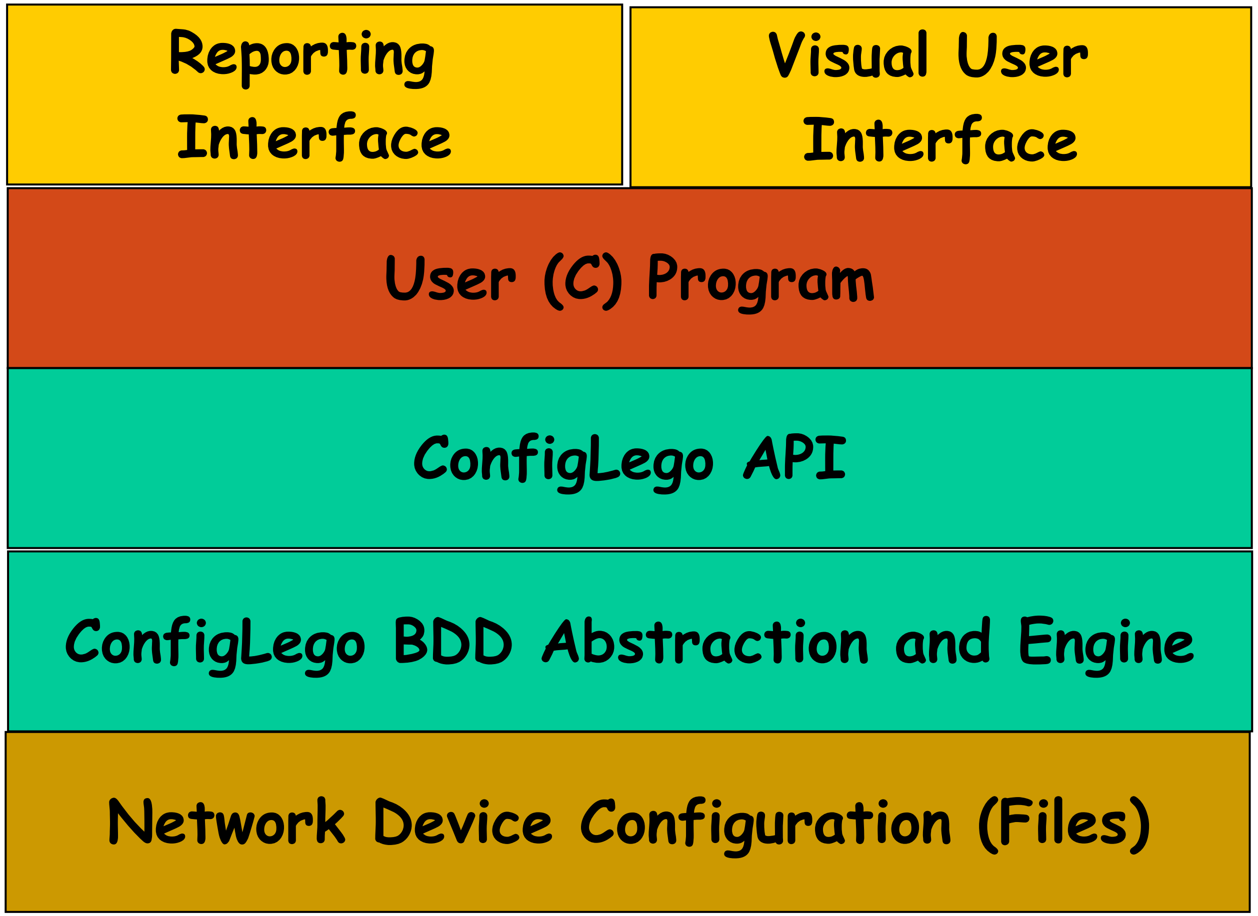
$$\phi_i = SAT_{(i=1,n)} \left(P_A^k \wedge \neg(\vee_{i=1}^n H_i) \wedge (\vee_{i=1}^n A_i) \wedge \neg(\wedge_{i=1,k-1} \phi_i)\right)$$

- Assume that variables $v_1, .. ,v_n$ are associated with cost $c_1, .. c_n$, what is the most *cost-effective* configuration changes to block attack scenarios $A_i,..,A_n$ without violating the requirements H

$$\phi_{minCost} = MinCostSAT(H \wedge \neg(\vee_{i=1}^n A_i))$$
$$P_A^k \leftarrow P_A^k \wedge \phi_{minCost}$$

  - To look for minimum number of config changes, assign the same cost as minCostSAT will minimize $C = \sum_{i=1}^n c_i * v_i$

ConfigLego



Reporting Interface
Visual User Interface
User (C) Program
ConfigLego API
ConfigLego BDD Abstraction and Engine
Network Device Configuration (Files)

ConfigLego Examples

Recap

- BDD can be used as primitives for configuration analysis
- Conflict/Inconsistency Analysis
- Fine-grain configuration optimization
- Configuration debugging and tracing
- Focus/limited configuration invitation and analysis