

Network Verification: Reflections from Electronic Design Automation (EDA)

Sharad Malik
Princeton University
MSR Faculty Summit: 7/8/2015

Microsoft Research
Faculty Summit
2015

\$4 Billion EDA industry
EDA Consortium
\$350 Billion Semiconductor Industry
World Semiconductor Trade Statistics
\$3 Trillion Electronics Industry
EDA Consortium



cādence®

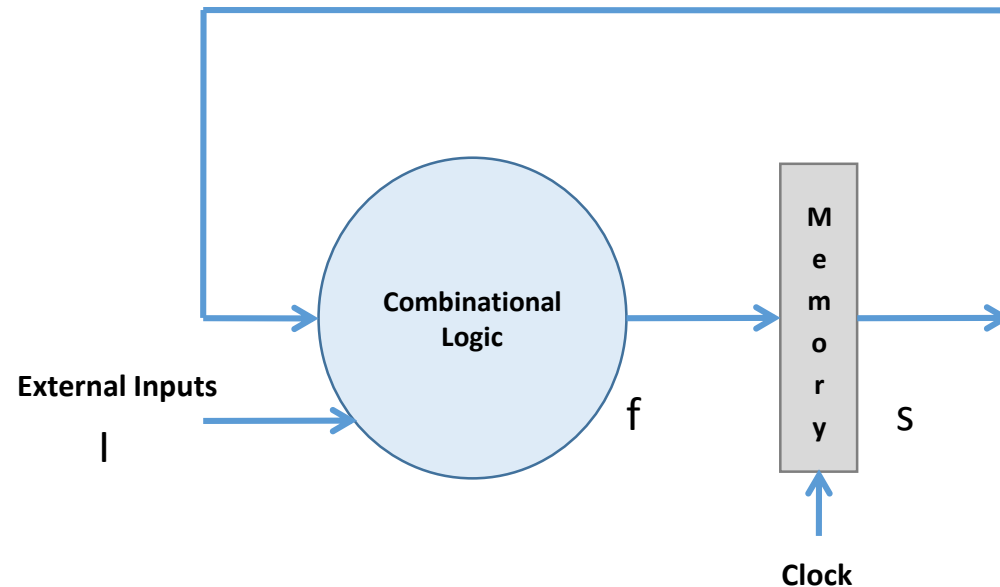
Mentor
Graphics®

SYNOPSYS®
Silicon to Software™

EDA: Design Tools *and* Design Methodology

- Interplay between design automation and design methodology
 - Design discipline
 - Synchronous design
 - State changes synchronously with clock

Disciplined Choice Restriction



$$s(t+1) = f(s(t), I)$$

Design discipline for
Network Design?

EDA: Design Tools *and* Design Methodology

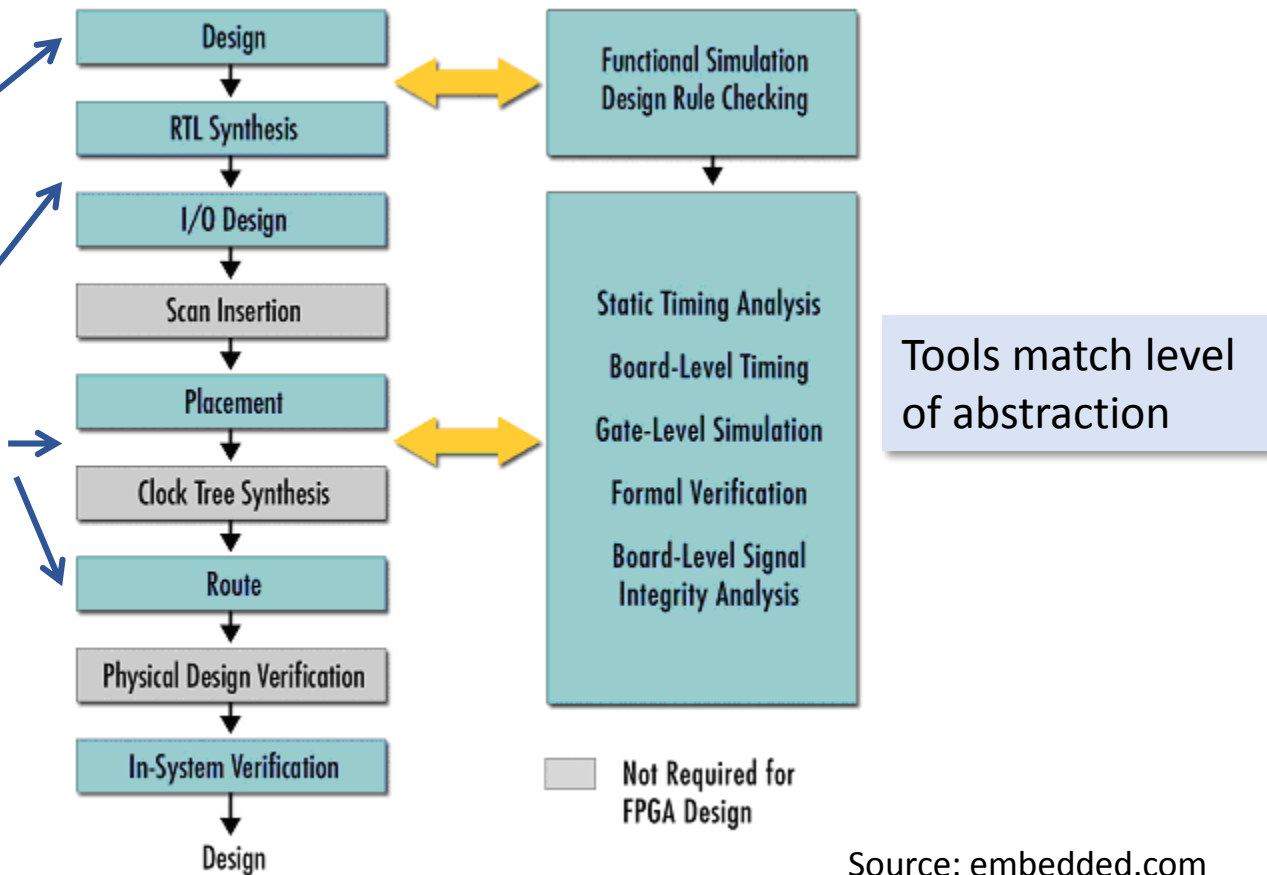
- Interplay between design automation and design methodology

- Design refinement

- Levels of abstraction

- Behavioral Design
 - Functions modifying state
 - Logic Design
 - Gates and their interconnections
 - Physical and Mask Design
 - Objects with 2D/3D coordinates

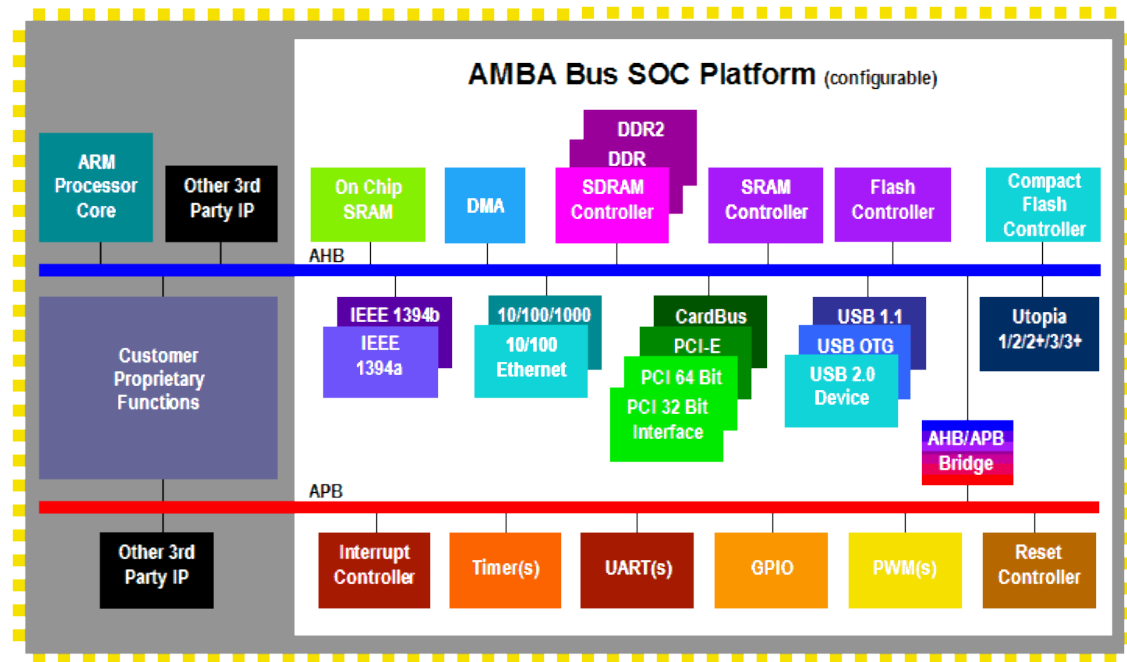
Design flows and abstractions for Network Design?



EDA: Design Tools *and* Design Methodology

- Interplay between design automation and design methodology
 - Design by composition
 - Standardized interfaces
 - Cell libraries
 - Memory interfaces
 - Bus interfaces

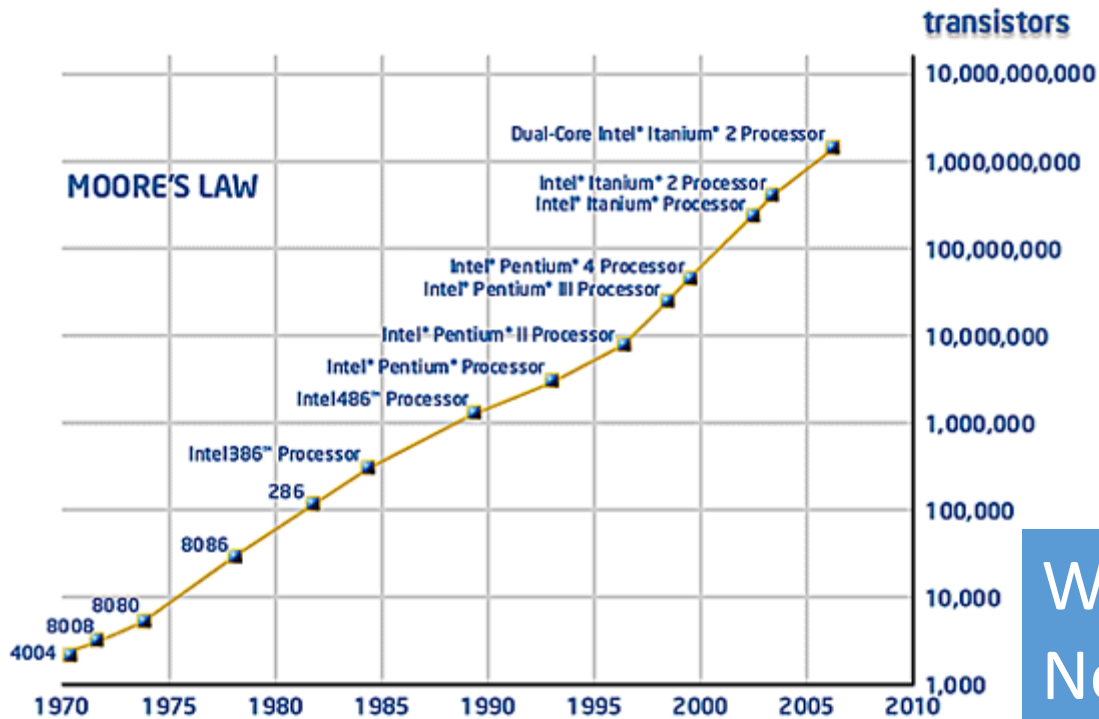
Standardized interfaces
for design by composition
in Network Design?



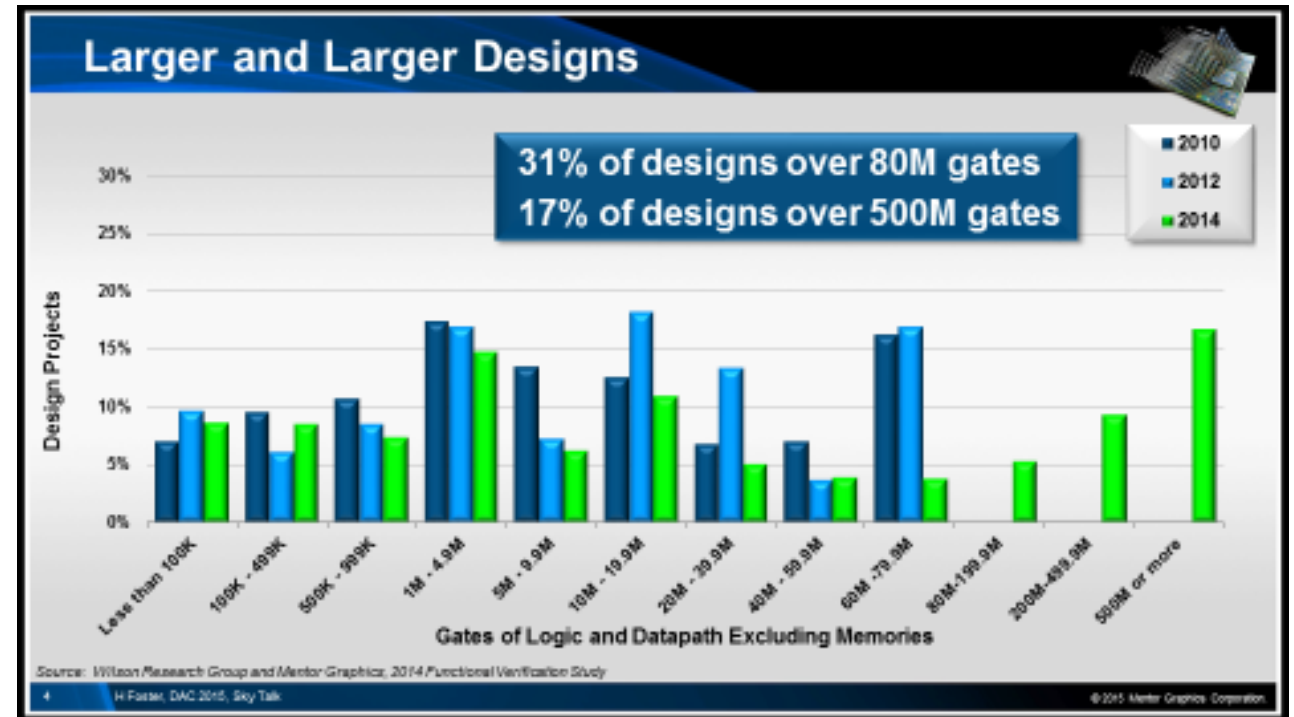
Source: arm.com

EDA Necessity

- Complexity



Source: intel.com



Source: Harry Foster, Mentor Graphics

Will growing network complexity make Network Design Automation (NDA) inevitable?

Hardware Verification Necessity

High cost of failure

- Need for first silicon success
 - High mask costs
- Product recalls
 - Intel Pentium FDIV Bug 1994
 - Total cost: \$475 million

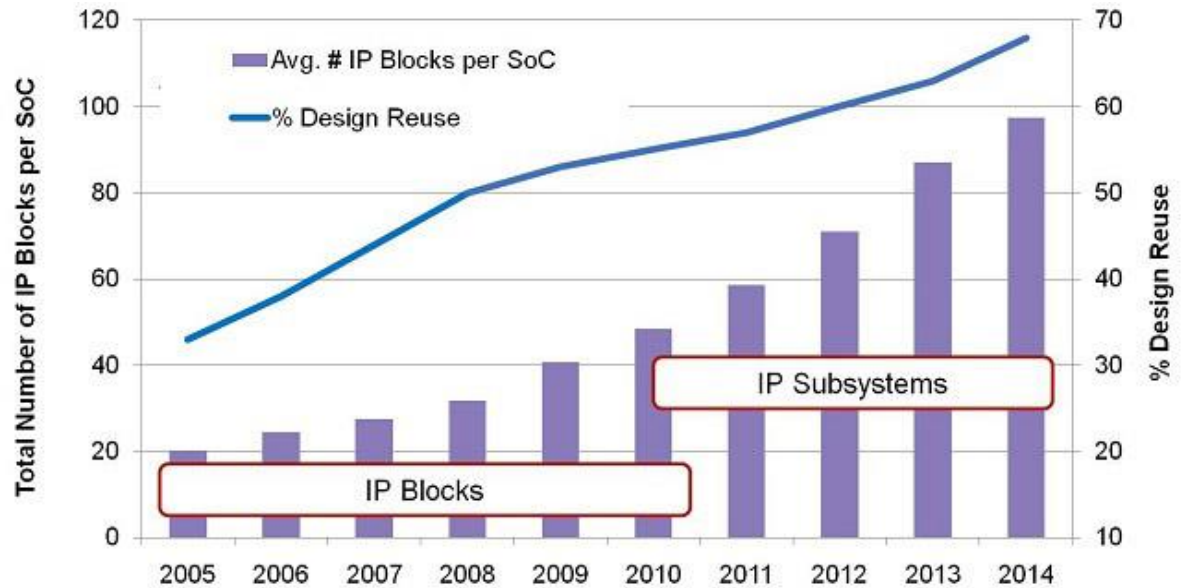


$$\frac{4195835}{3145727} = 1.333839068900237689$$

Down time and security breach costs compelling for Network Verification

EDA Value Proposition

- Design scalability
- Design diversity
 - Diversity of components
 - Specialized functions
 - Intellectual Property (IP)
 - Diversity of Parts
 - Application Specific Integrated Circuits
 - Systems-on-a-Chip
 - Integrate IP
- Design optimization
 - Reduce part cost
 - Enable new functionality
 - Push the power-cost-performance frontier



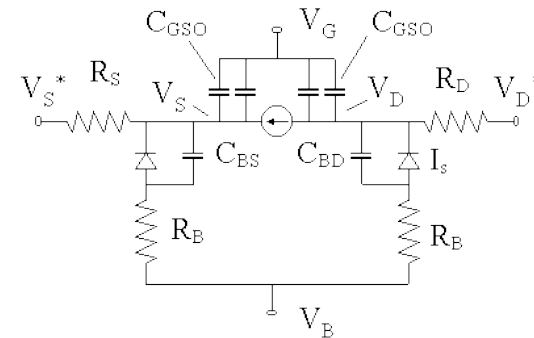
Source: Semico, October 2010

Benefits from scalability, diversity, cost reduction, novel functionality compelling enough for NDA?

EDA Evolution

- Models
 - Spice
 - Equations to model semiconductor devices

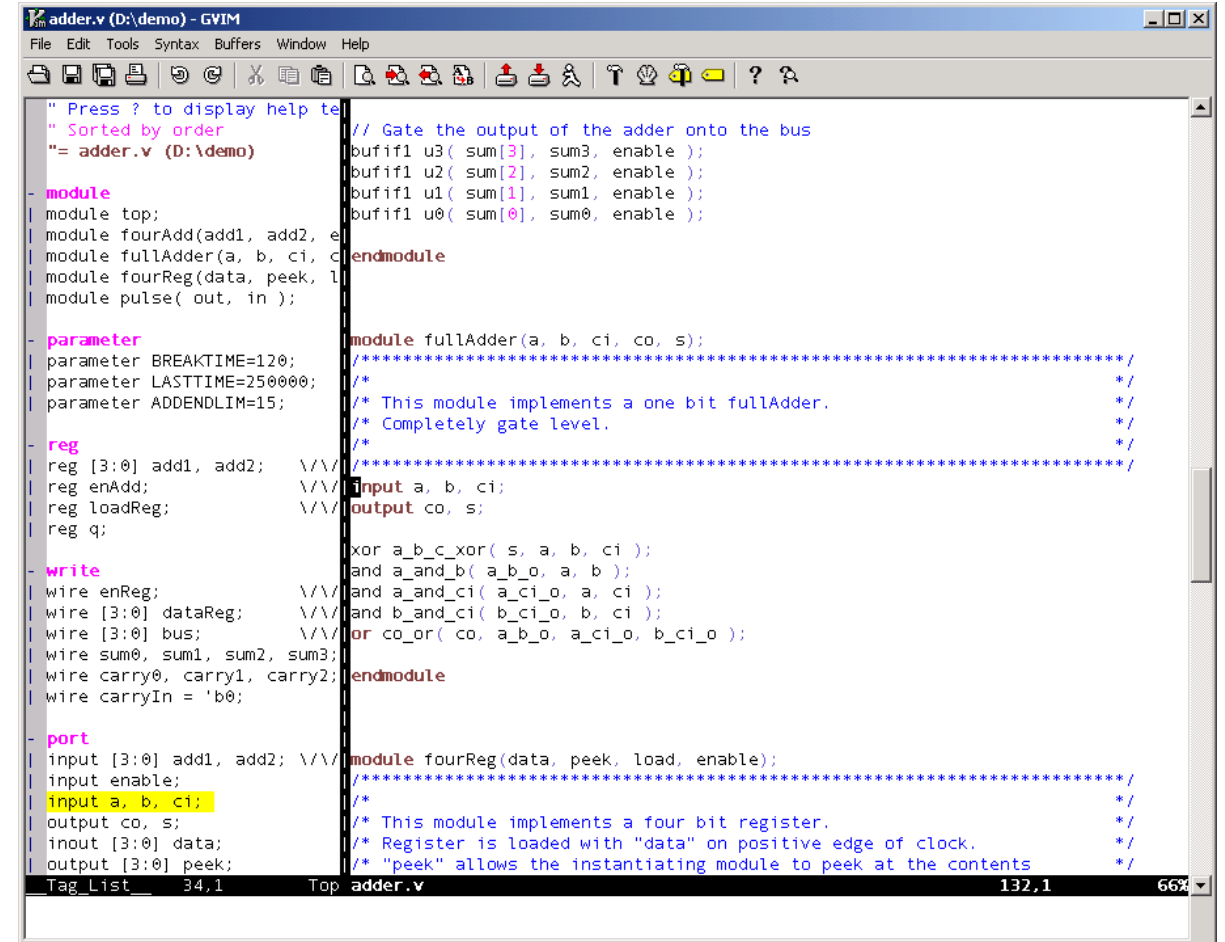
SPICE variable	Equation
TOX	$TOX = t_{ox}$
KP	$KP = \mu C_{OX}$
VTO	$VTO = V_{FB} + 2\phi_F + \frac{\sqrt{2\epsilon_s q N_a (2\phi_F)}}{C_{OX}}$
GAMMA	$GAMMA = \gamma = \frac{\sqrt{2\epsilon_s q N_a}}{C_{OX}}$
NSUB	$NSUB = N_d \text{ or } N_a$
U0	$U0 = \mu$
LAMBDA	$LAMBDA = \lambda$
VMAX	$VMAX = v_{sat}$



Source: ecee.colorado.edu

EDA Evolution

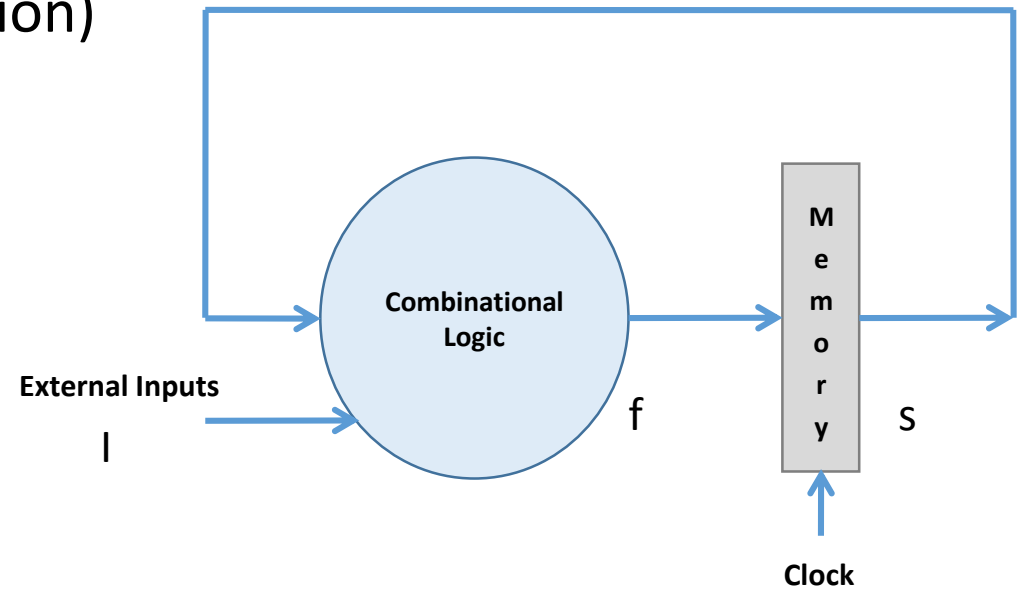
- Models
 - Spice
 - Equations to model semiconductor devices
 - Verilog
 - Design specification with underlying model



```
adder.v (D:\demo) - GVIM
File Edit Tools Syntax Buffers Window Help
[Icons]
" Press ? to display help te
" Sorted by order
"= adder.v (D:\demo)
- module
| module top;
| module fourAdd(add1, add2, e
| module fullAdder(a, b, ci, c
| module fourReg(data, peek, l
| module pulse( out, in );
- parameter
| parameter BREAKTIME=120;
| parameter LASTTIME=250000;
| parameter ADDENDLIM=15;
- reg
| reg [3:0] add1, add2; \/\
| reg enAdd; \/\
| reg loadReg; \/\
| reg q;
- write
| wire enReg; \/\
| wire [3:0] dataReg; \/\
| wire [3:0] bus; \/\
| wire sum0, sum1, sum2, sum3;
| wire carry0, carry1, carry2;
| wire carryIn = 'b0;
- port
| input [3:0] add1, add2; \/\
| input enable;
| input a, b, ci;
| output co, s;
| inout [3:0] data;
| output [3:0] peek;
// Gate the output of the adder onto the bus
bufif1 u3( sum[3], sum3, enable );
bufif1 u2( sum[2], sum2, enable );
bufif1 u1( sum[1], sum1, enable );
bufif1 u0( sum[0], sum0, enable );
endmodule
- parameter
| module fullAdder(a, b, ci, co, s);
| *****
| /*
| /* This module implements a one bit fullAdder.
| /* Completely gate level.
| /*
| /* *****
| input a, b, ci;
| output co, s;
| xor a_b_c_xor( s, a, b, ci );
| and a_and_b( a_b_o, a, b );
| and a_and_ci( a_ci_o, a, ci );
| and b_and_ci( b_ci_o, b, ci );
| or co_or( co, a_b_o, a_ci_o, b_ci_o );
endmodule
- parameter
| module fourReg(data, peek, load, enable);
| *****
| /*
| /* This module implements a four bit register.
| /* Register is loaded with "data" on positive edge of clock.
| /* "peek" allows the instantiating module to peek at the contents
| /* *****
| input [3:0] data;
| output [3:0] peek;
Tag_List_ 34,1 Top adder.v 132,1 66%
```

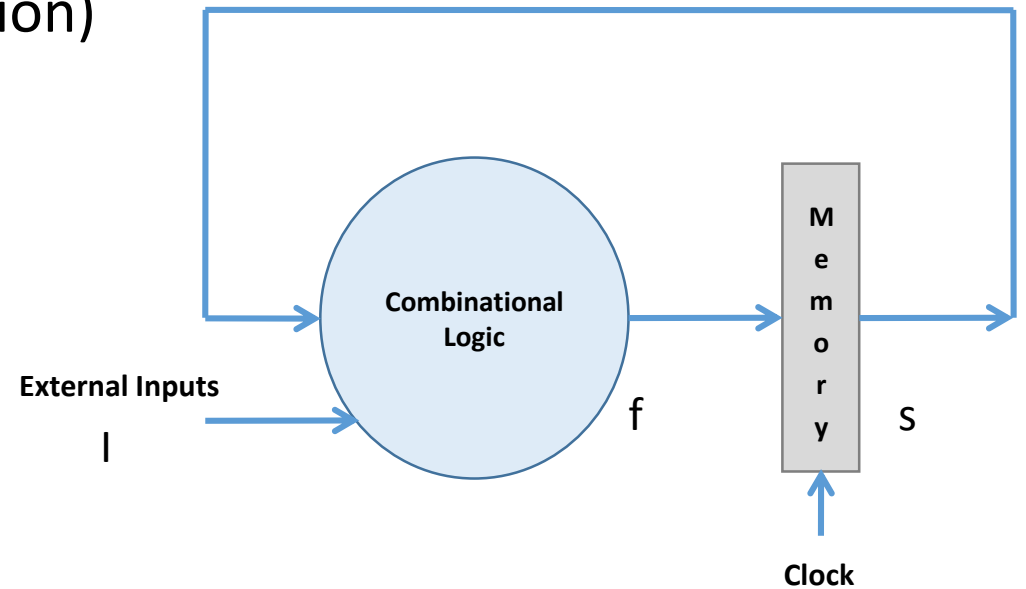
EDA Evolution

- Models
 - Spice
 - Equations to model semiconductor devices
 - Verilog (Specification and Models connection)
- Analysis
 - Timing analysis
 - Functional analysis (verification)



EDA Evolution

- Models
 - Spice
 - Equations to model semiconductor devices
 - Verilog (Specification and Models connection)
- Analysis
 - Timing analysis
 - Functional analysis (verification)
- Optimization



EDA Evolution

- Models
 - Spice
 - Equations to model semiconductor devices
 - Verilog (Specification and Models connection)
- Analysis
 - Timing analysis
 - Functional analysis (verification)
- Optimization
- Synthesis
 - Creating optimized designs from specifications

```
adderv (D:\demo) - GVIM
File Edit Tools Syntax Buffers Window Help
Press ? to display help text
Sorted by order
*= adderv.v (D:\demo)
// Gate the output of the adder onto the bus
bufif1 u2( sum[3], sum3, enable );
bufif1 u1( sum[2], sum2, enable );
bufif1 u1( sum[1], sum1, enable );
bufif1 u0( sum[0], sum0, enable );

module top;
module fourAdd(add1, add2, enable);
module fullAdder(a, b, c1, c0);
module fourReg(data, peek, load, enable);
module pulse(out, in);

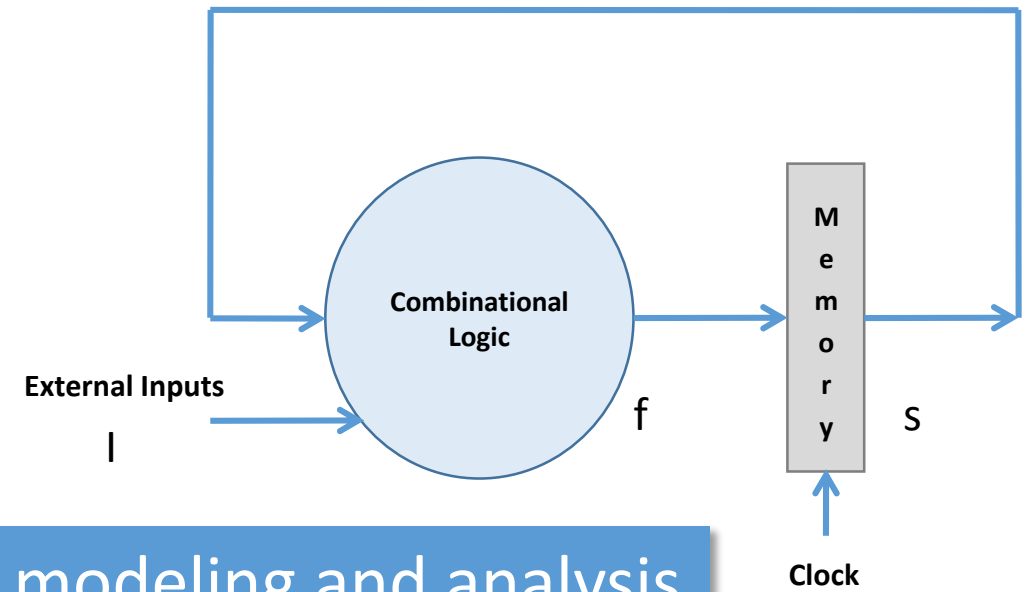
parameter
parameter BREAKTIME=120;
parameter LASTTIME=250000;
parameter ADDENDIM=15;

reg [3:0] add1, add2; //V
reg enAdd; //V
reg loadReg; //V
reg q;

write
wire enReg; //V
wire [3:0] dataReg; //V
wire [3:0] bus; //V
wire sum0, sum1, sum2, sum3;
wire carry0, carry1, carry2;
wire carryIn = 'b0;

port
input [3:0] add1, add2; //V
input enable;
input a, b, c1;
output co, s;
output [3:0] data;
output [3:0] peek;

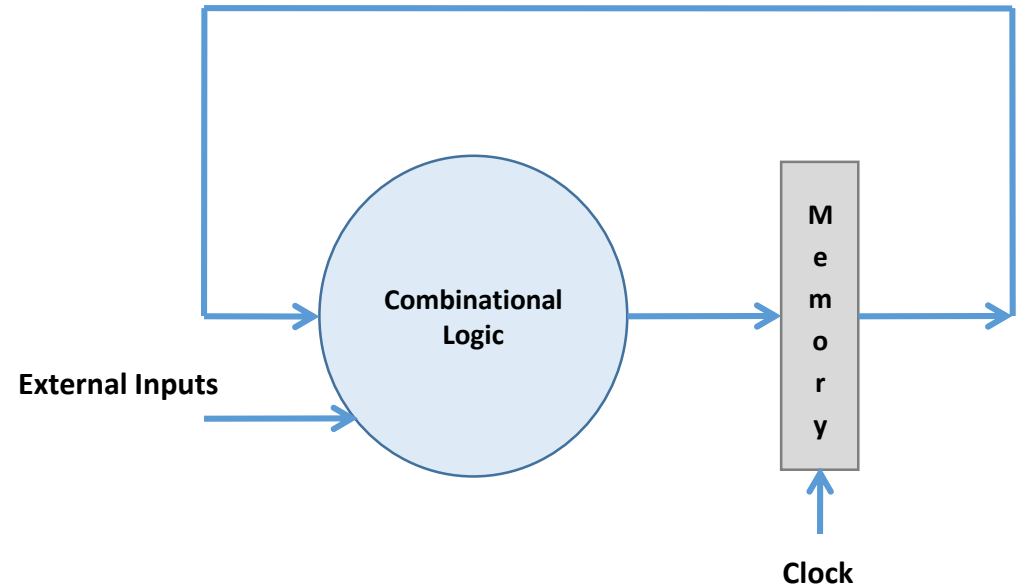
Tag_List 34.1 Top adderv.v 132.1 66%
```



Effective modeling and analysis for enabling NDA?

Analysis Capability Impacts Design Methodology

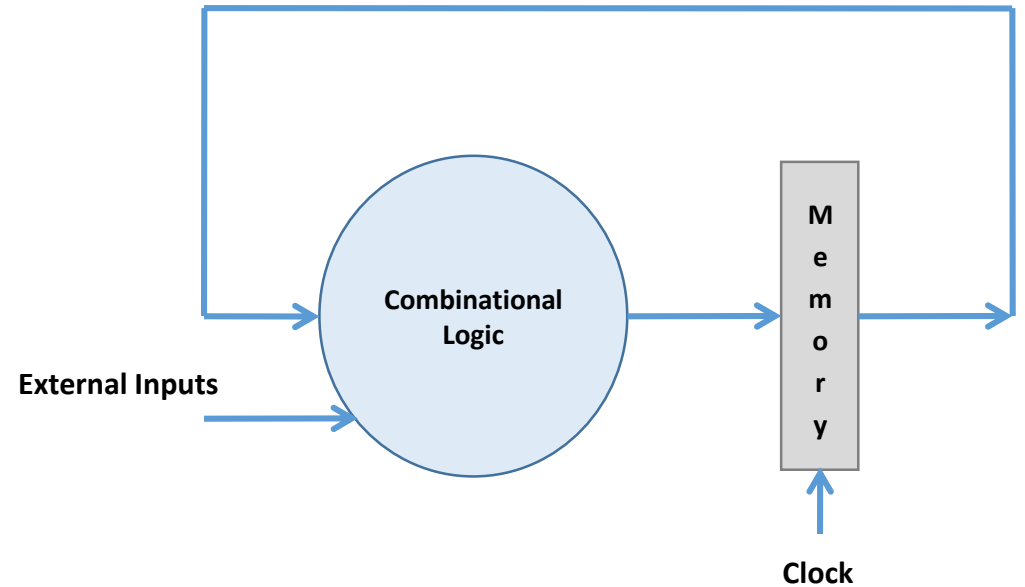
- Separation of Concerns
 - Separating timing and functional verification
 - Static timing analysis ignores functionality
 - Functional verification ignores timing
 - Driver for synchronous design methodology



Maximizing separation of concerns in Network Design?

Analysis Capability Impacts Design Methodology

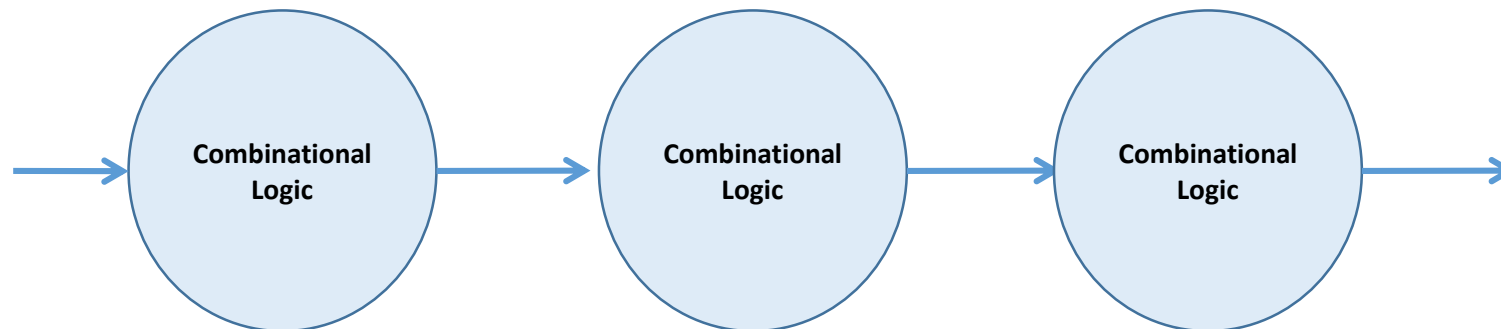
- Design Verification
 - Reasoning about combinational logic easier than reasoning about sequential logic
 - Designs obey initial register placements
 - state definition
 - Implementation verification reduces to combinational equivalence checking



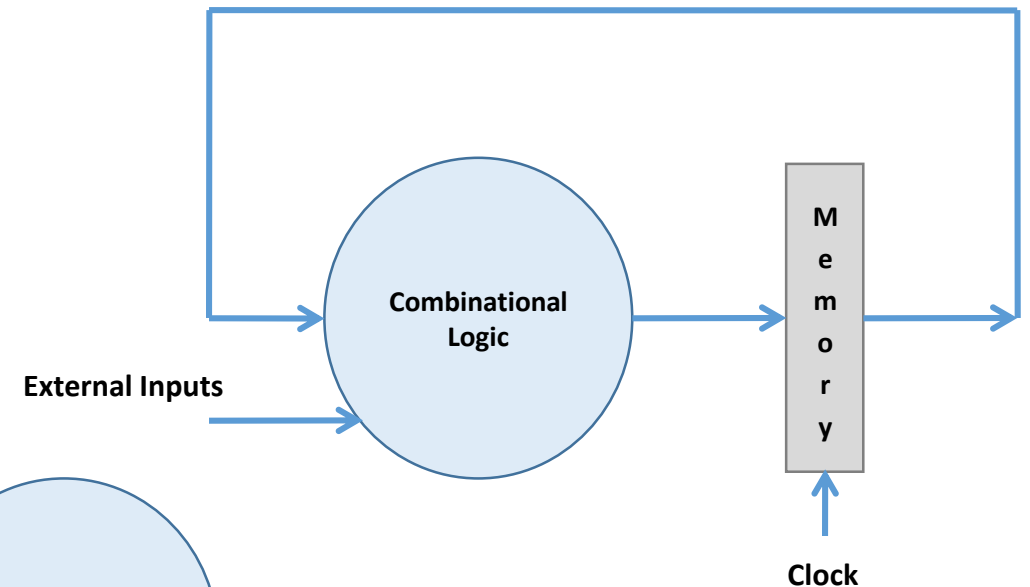
Verification Driven Design Discipline

Analysis Capability Impacts Verification Methodology

- Design Verification
 - Reasoning about combinational logic easier than reasoning about sequential logic
 - Bounded model checking easier than unbounded model checking



k-cycle verification



Analysis Capability Impacting
Verification Methodology

SAT Based Verification of Network Data Planes

(joint work with Shuyuan Zhang)

Motivation: Avoid State Space Exploration

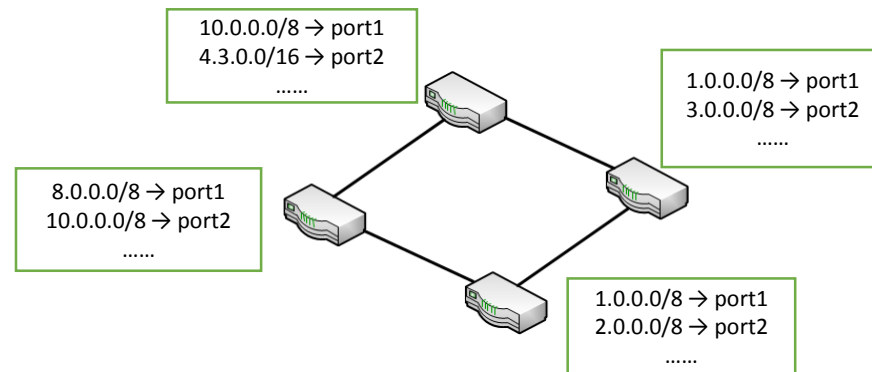
- Large State Space
 - Packet Header Size \approx 100s of bits
 - MAC address: 48 bits
 - IP address: 32 bits
 - TCP port: 16 bits
 - VLAN, In port, Ethernet type...
 - Network Size
 - Tens to thousands of switches
 - Each switch generally has 1k~5k rules
 - Buffers...
 - Concurrency
 - Switches operate in parallel
 - Large number of packet interleavings

Model Checking vs. Propositional Logic with SAT

PSPACE-Complete vs. NP-Complete

Snapshot Verification

- Verify the static network state
 - A snapshot of a dynamic system
 - A single SDN rule configuration
 - Ignore network performance

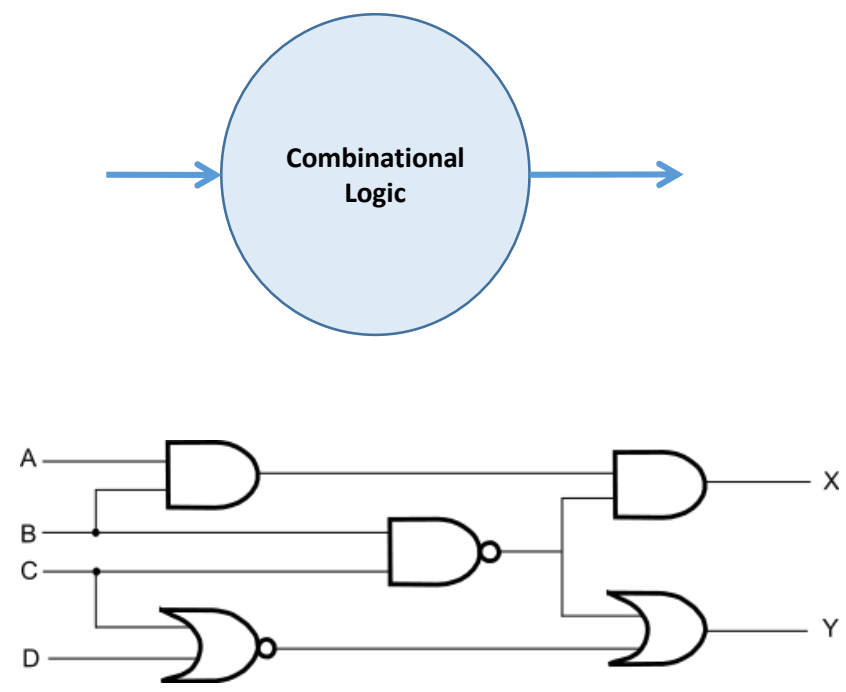
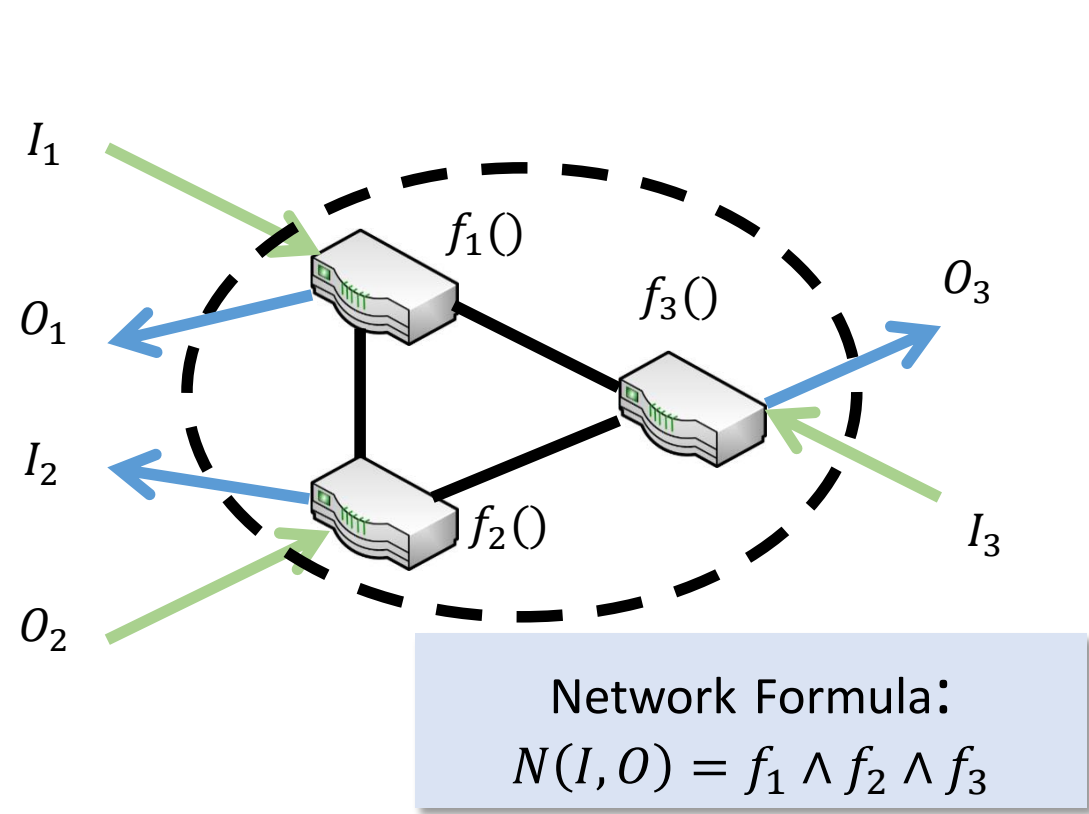


Static network switch state ✓
Packet header state?
Network state due to interleavings?

- Network state change (rule deletion/addition/change at a switch)^[1]
 - Tens of events per second
- Packet arrival rate
 - Millions of arrivals per second

[1] Gude, N., Koponen, T., Pettit, J., Pfa, B., Casado, M., McKeown, N., Shenker, S.: “Nox: towards an operating system for networks,” SIGCOMM 2008

Data Plane as a Logic Circuit

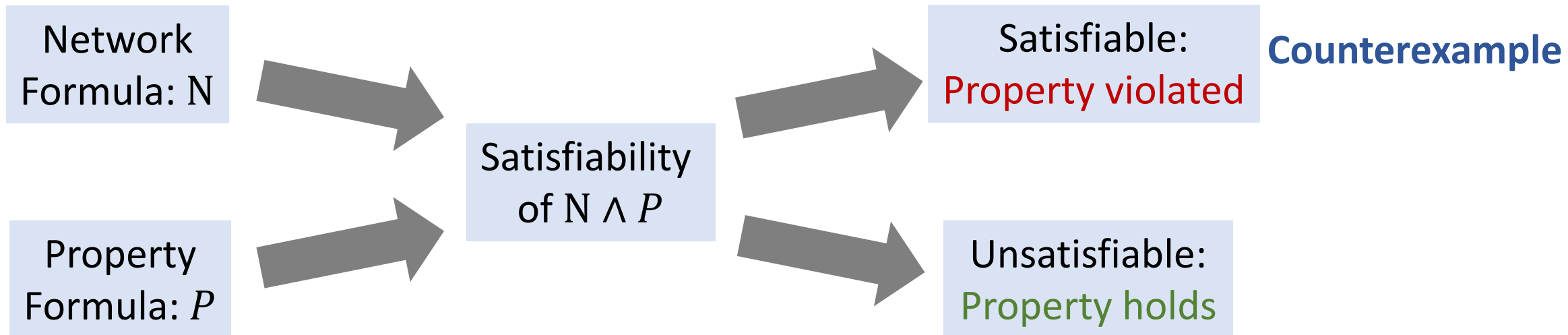


- Model it as a combinational logic circuit?
 - Outputs and signals are functions of only the present value of the inputs

SAT Based Property Verification

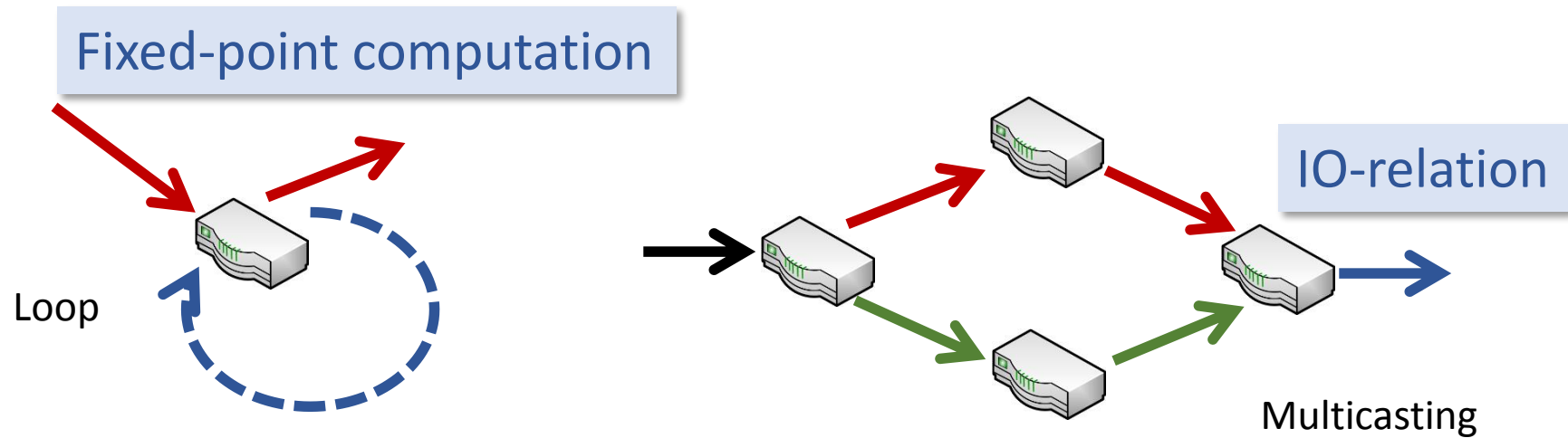
- Property Formula

- Encode negation of the property: finding counter examples
 - Example: Check the reachability from A to B
 - Property Formula: conditions for a packet to reach places other than B



Modeling/Analysis Challenge

- Even for a single packet entering a network, a link may see multiple packets

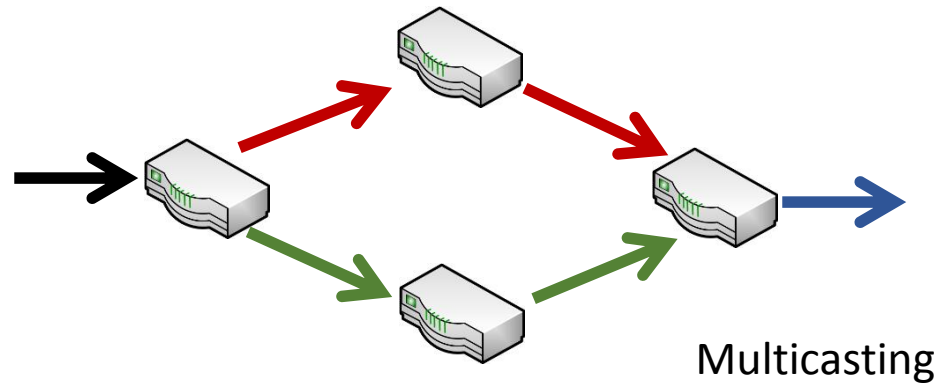
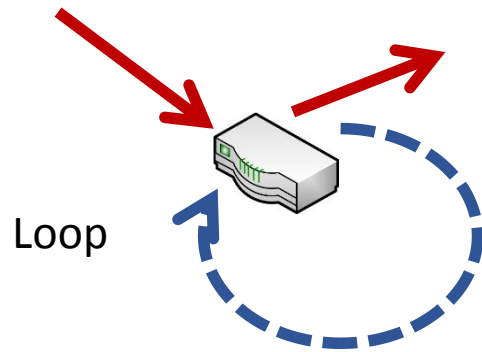


- Switch output not a combinational function of its inputs

Need to store sets of values

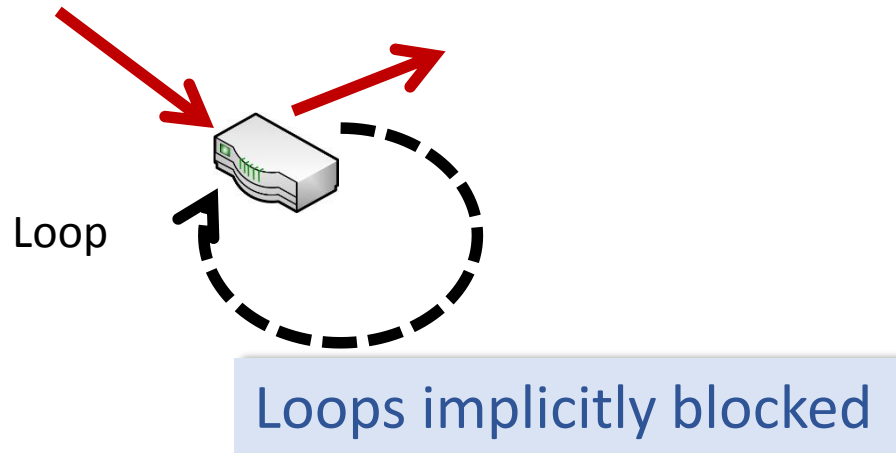
Adapting Modeling/Analysis

- Limit packet flow to a *single path for a single packet* through the network

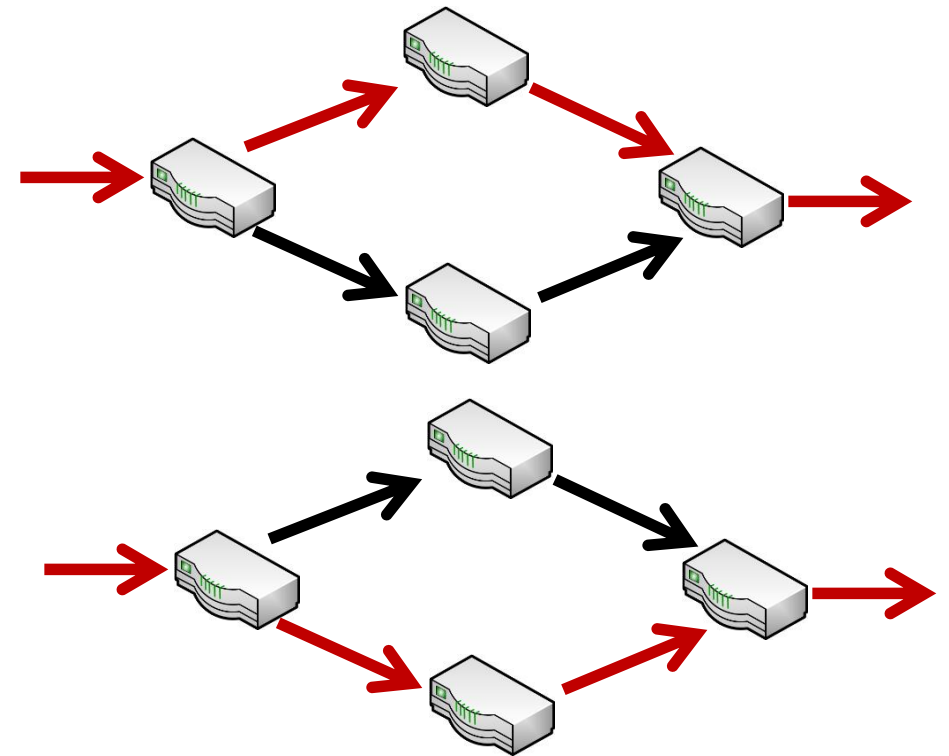


Adapting Modeling/Analysis

- Limit packet flow to a *single path for a single packet* through the network



- Captures only part of the network behavior
- What good is this?

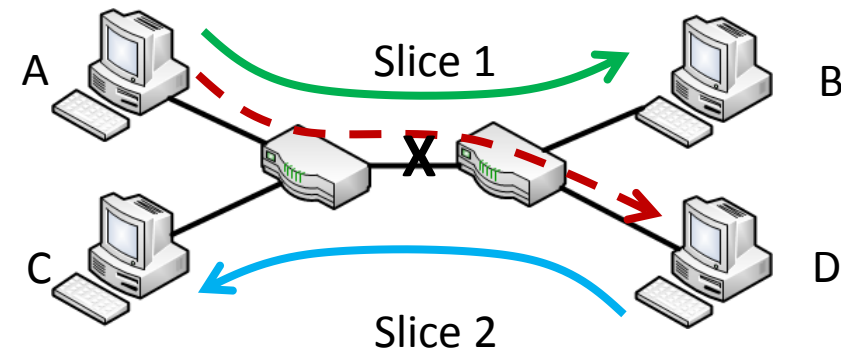
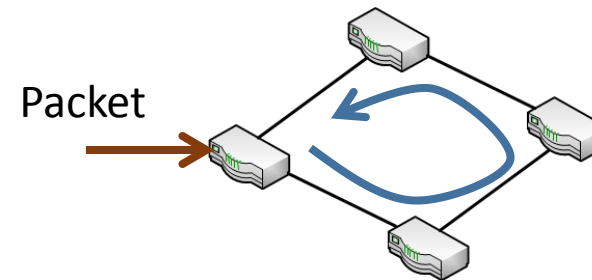
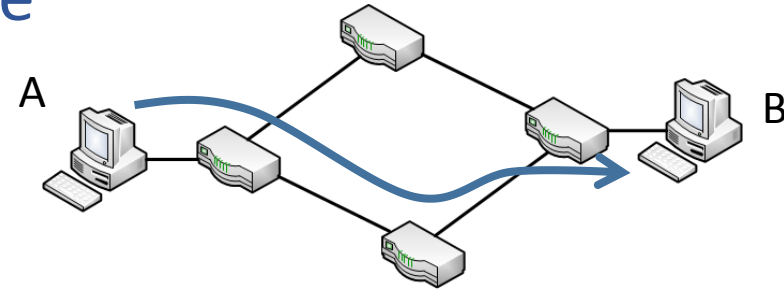


Goal: Counterexamples for Property Failures

Single Path Single Packet Counterexample

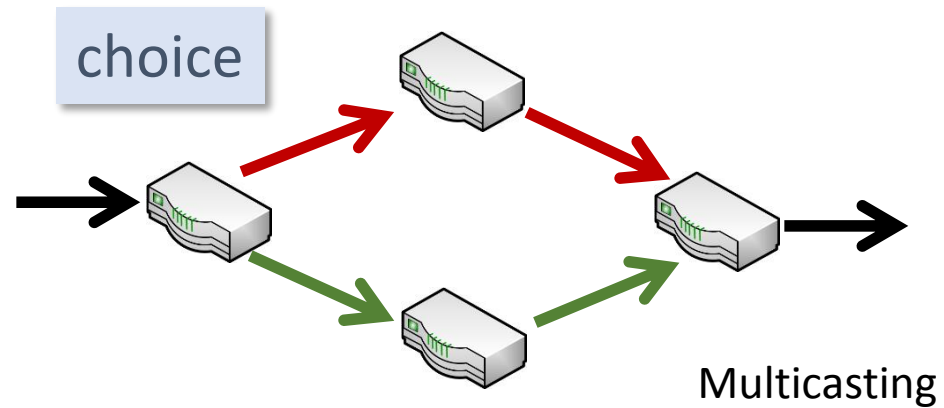
Suffices for

- Functional Properties:
 - Reachability checking
 - Waypointing
 - Blacklisting
- Functional/Performance Properties:
 - Forwarding loop
- Security Properties:
 - Slice isolation
 - virtualization context



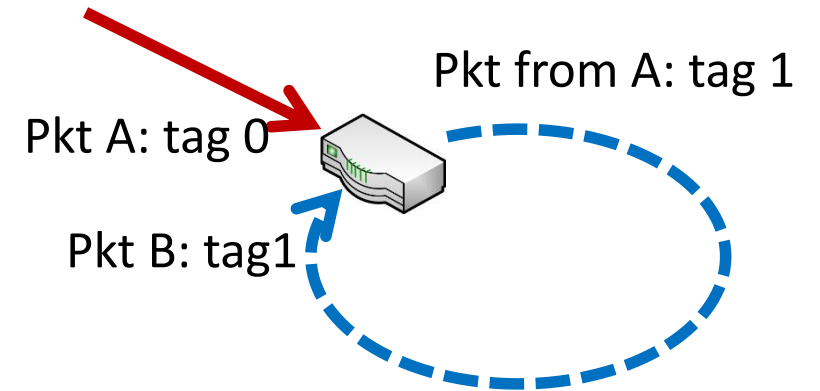
Adapting Modeling/Analysis

- Non-deterministically select one of the paths
 - choice variable
- Solver explores all possibilities for counterexample



Adapting Modeling/Analysis

- Extra tag bit tracks looping
 - Packets enter the network with tag 0
 - Switch with two incoming packets:
 - One of the two packets has looped
 - Switch selects packet with tag 0 for forwarding
 - The tag of output packet is 1
 - Looping packet is blocked
 - Minimally unroll to check for k-times-looping
- Packet loops iff there exists a switch with two incoming packets
 - Easy check for packet looping



Avoid maintaining full path history

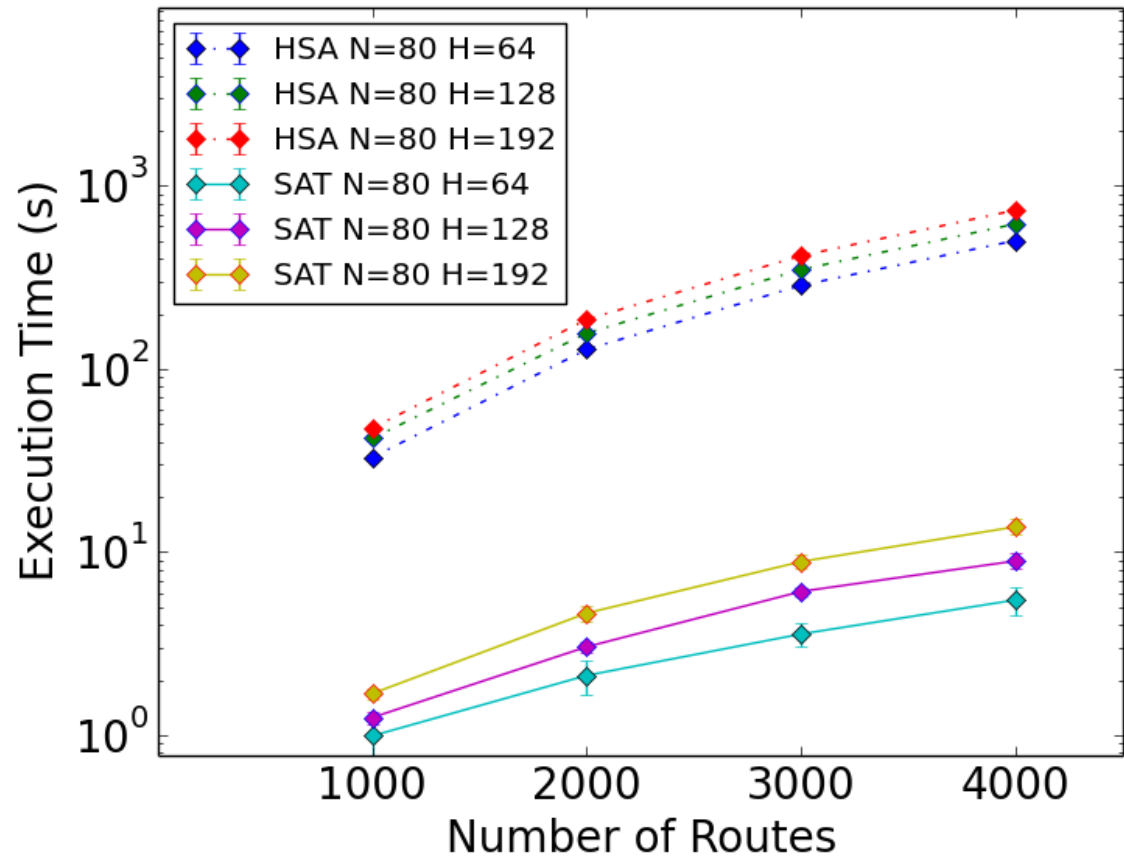
Experimental Results

Setup

- SAT solver: Minisat
- Stanford backbone network
 - 16 routers with full network functions (VLAN, ACL, ...)
 - $\approx 15,000$ rules
 - 129 seconds to find a forwarding loop
 - Header Space Analysis (HSA): 758 seconds
 - Uses Ternary Symbolic Simulation
- Synthetic benchmarks for scalability experiments
 - Fat tree topology
 - Shortest path routing
 - Depth-first-search to generate matching rules
 - Vary
 - # of switches: N
 - # of routes: P
 - # of packet header bits: H

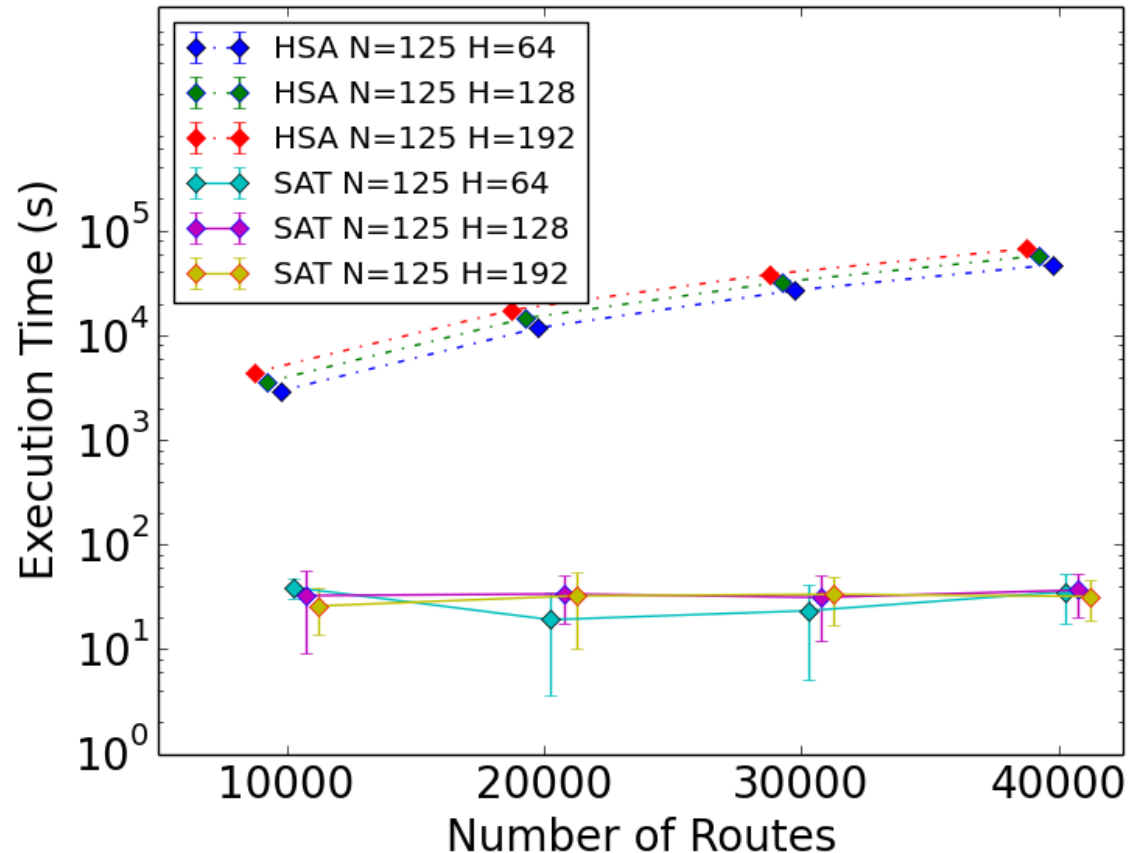
Experimental Results

- Property
 - Forwarding loop check
- Setup
 - Vary
 - # Routes
 - # of Header bits
 - HSA: Header Space Analysis
 - SAT: SAT-based method
- Observations
 - Sub-exponential growth with number of routes
 - Low dependence on header size



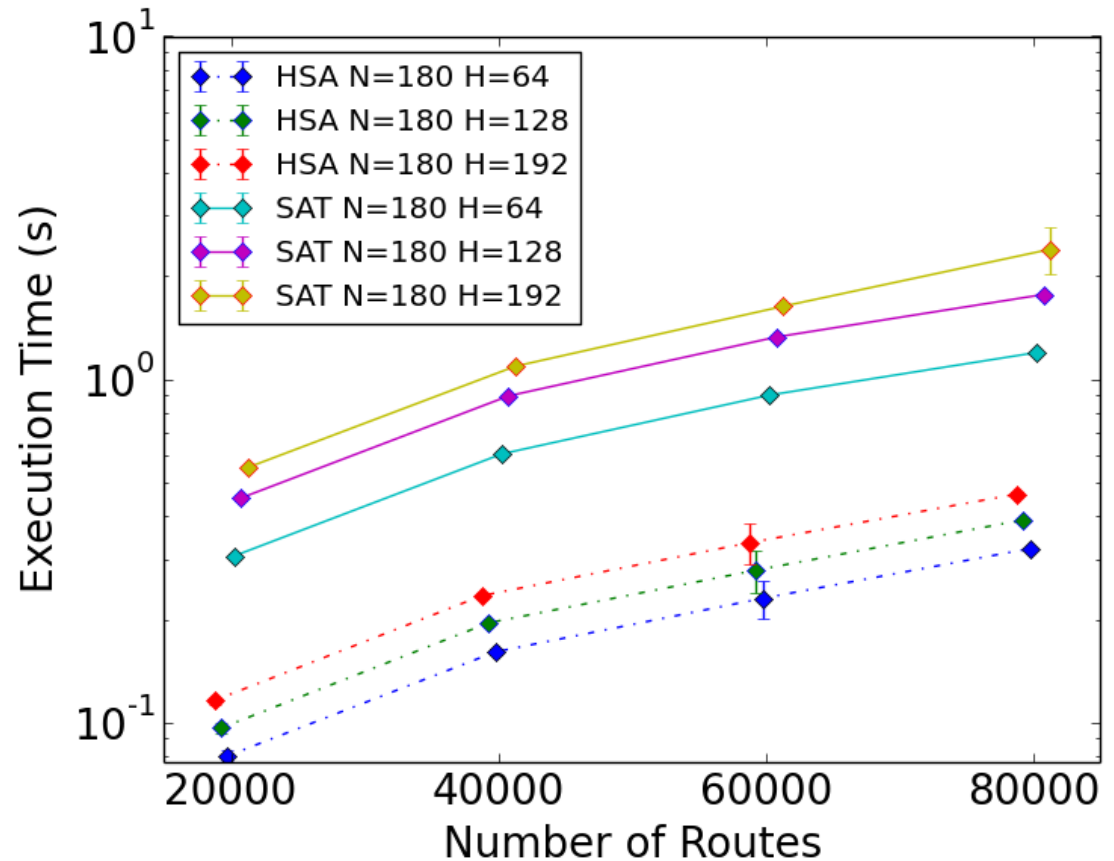
Experimental Results

- Property
 - Forwarding loop check
- Setup
 - Vary
 - # Routes
 - # of Header bits
 - HSA: Header Space Analysis
 - SAT: SAT-based method
- Observations
 - Sub-exponential growth with number of routes
 - Low dependence on header size



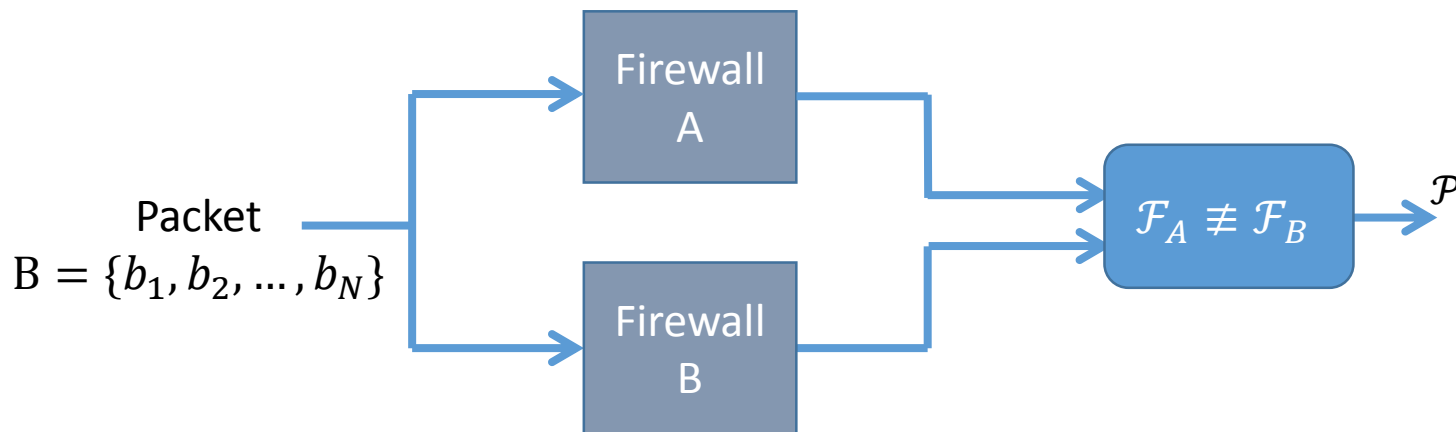
Experimental Results

- Property
 - Reachability check
- Setup
 - Vary
 - # Routes
 - # of Header bits
 - HSA: Header Space Analysis
 - SAT: SAT-based method
- Observations
 - Sub-exponential growth with number of routes
 - Low dependence on header size



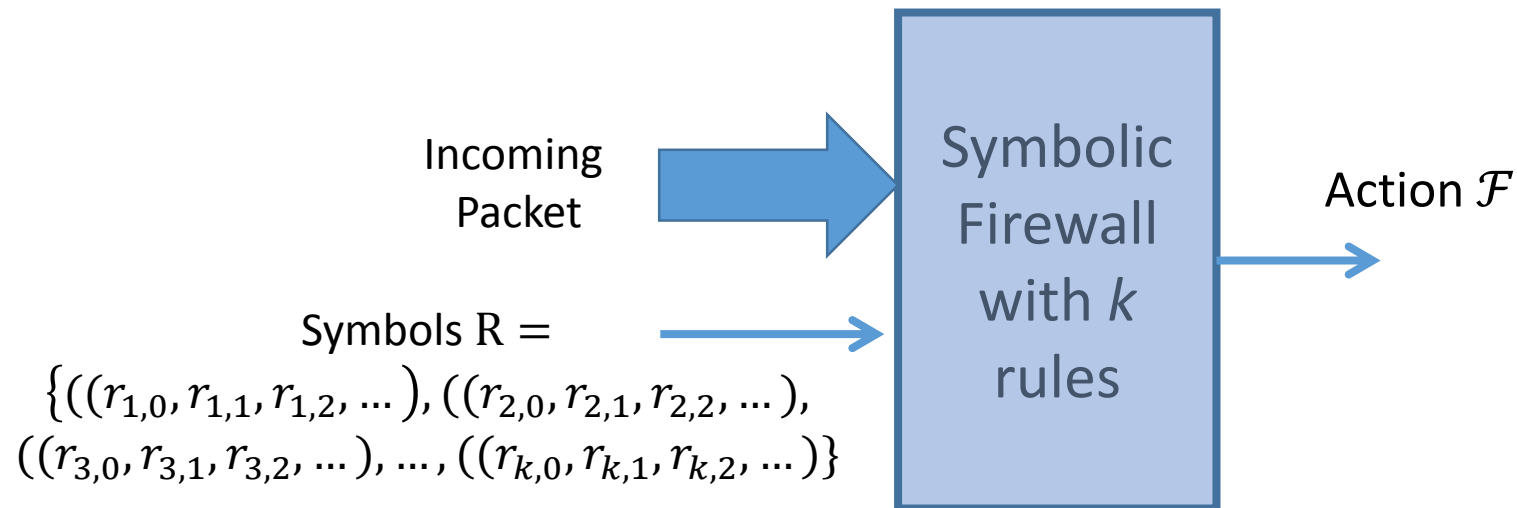
From Analysis to Synthesis: Firewall Case Study

- Firewall Equivalence Checking
 - $\mathcal{P} = \mathcal{F}_A \not\equiv \mathcal{F}_B$
 - \mathcal{P} satisfiable \rightarrow not equivalent
 - \mathcal{P} unsatisfiable \rightarrow equivalent



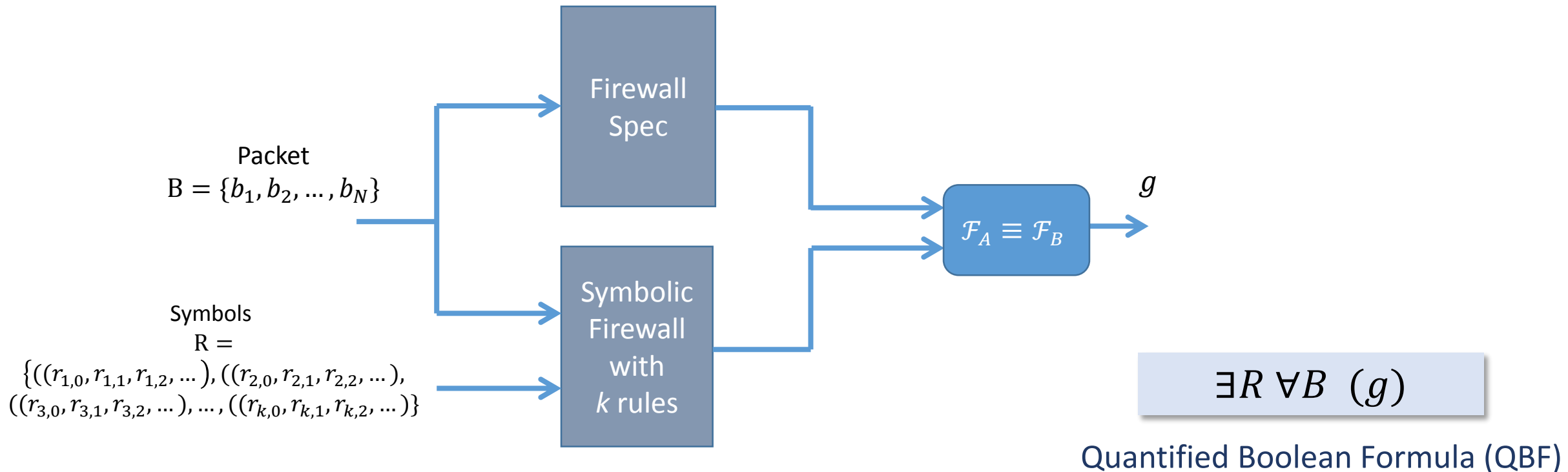
Firewall Synthesis

- Firewall Synthesis
 - Firewall with the fewest rules for a given specification
- Symbolic Firewalls
 - Represents all firewalls with k rules



- ▶ Each assignment to R specifies one firewall

Firewall Synthesis



- Find an R , if one exists, such that for all B , g holds
- Binary search for minimum k
- Practical QBF (and special purpose) solvers do not scale well

In thinking about Network Design Automation...

Design discipline for Network Design?

Design flows, abstractions and interfaces for Network Design?

Effective modeling and analysis to enable NDA evolution?

Maximizing separation of concerns in Network Design?

Analysis capabilities influencing Network Design/Verification methodology?