Microsoft

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

DEI SVB NVMINE VIGET

cadence®

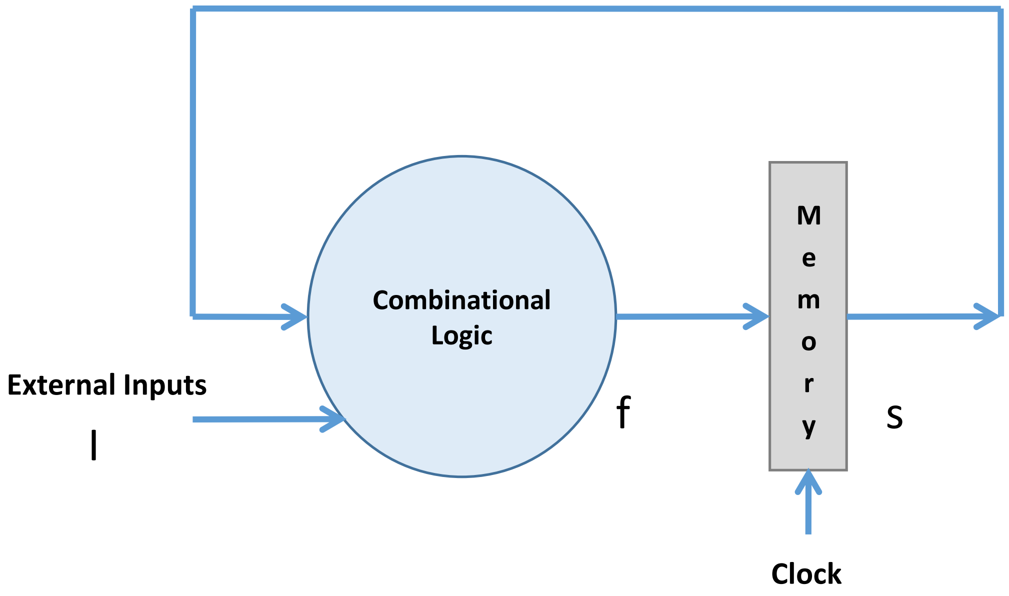
Mentor Graphics®

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

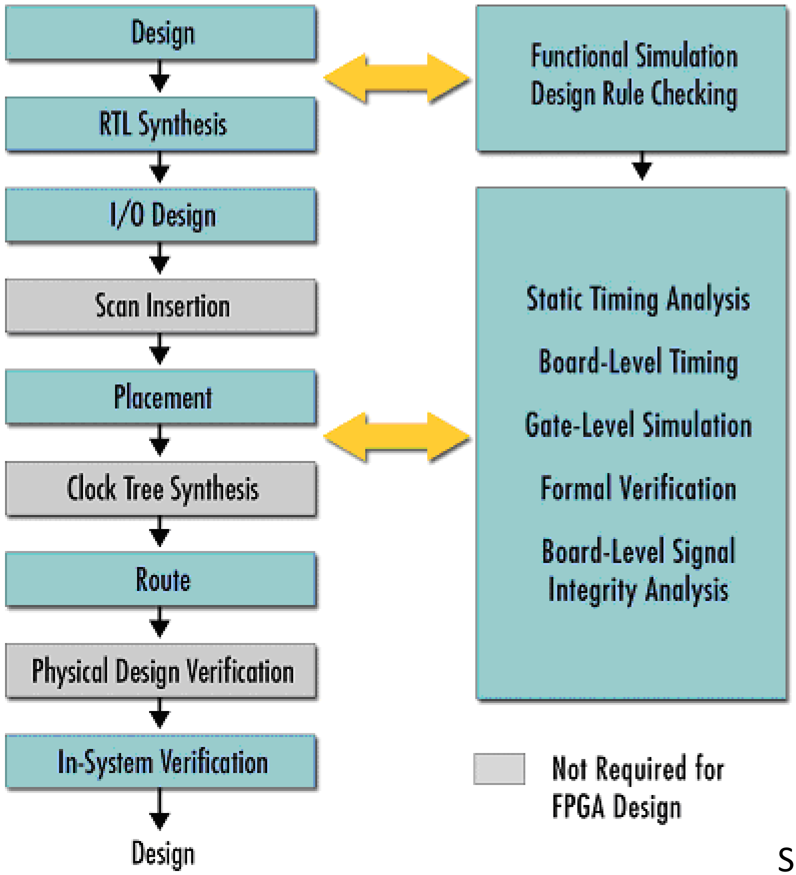


$s(t+1) = f(s(t), I)$

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



Tools match level of abstraction

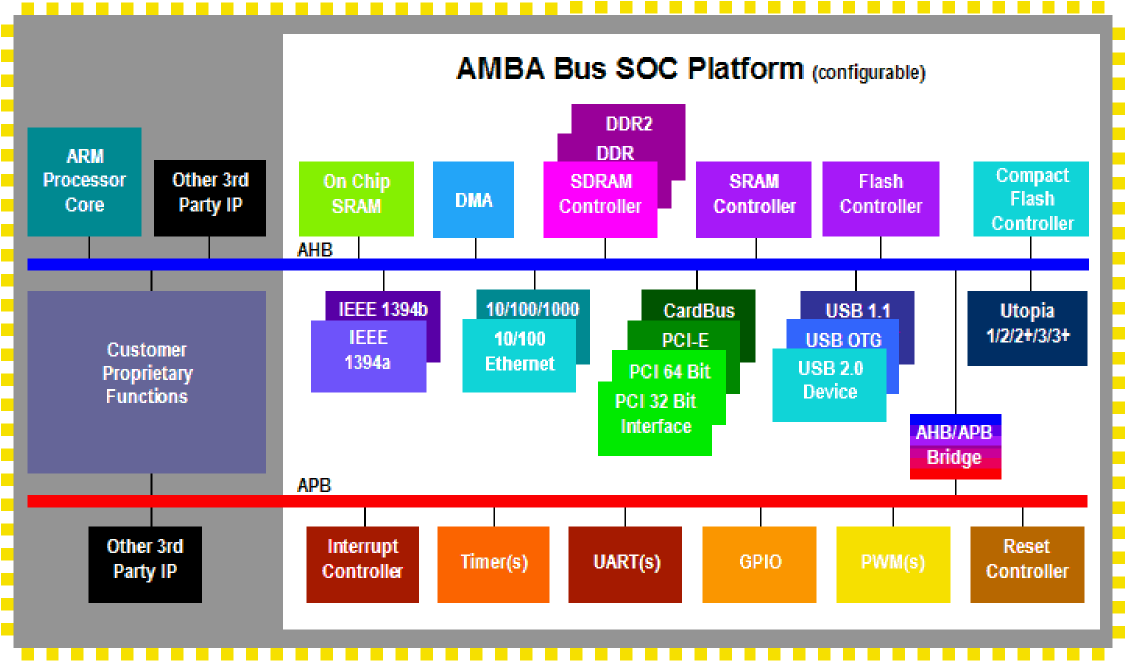
Design flows and abstractions for Network Design?

Source: embedded.com

EDA: Design Tools ***and*** Design Methodology

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

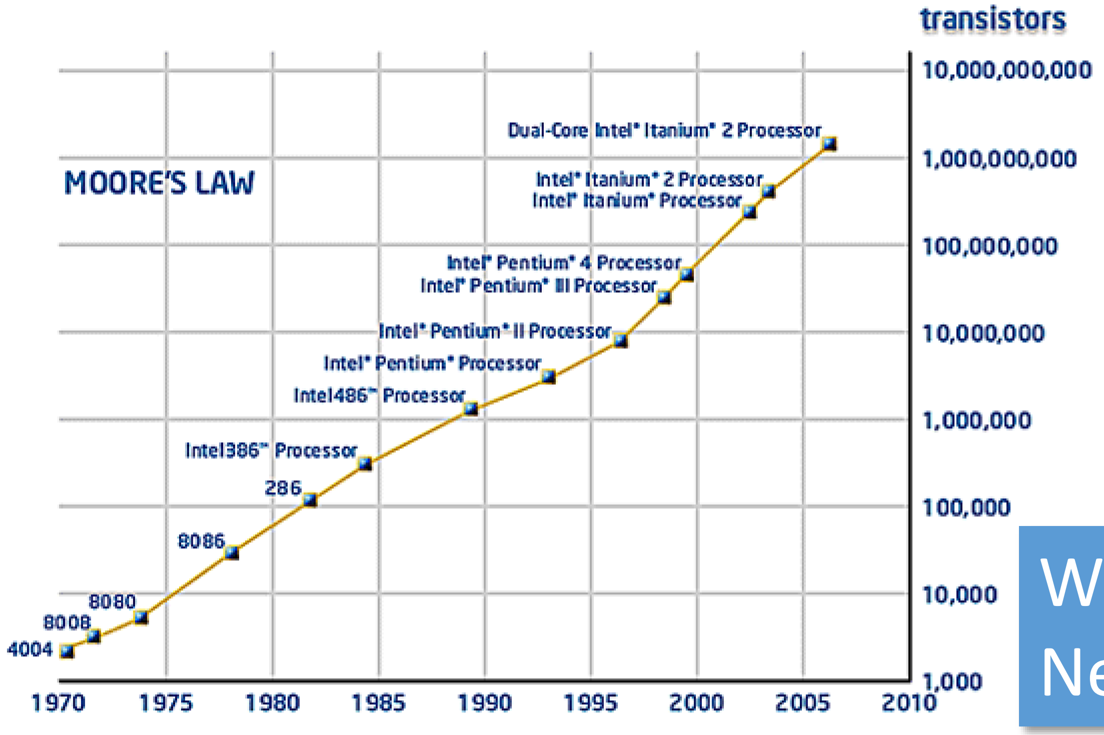
Standardized interfaces
for design by composition
in Network Design?



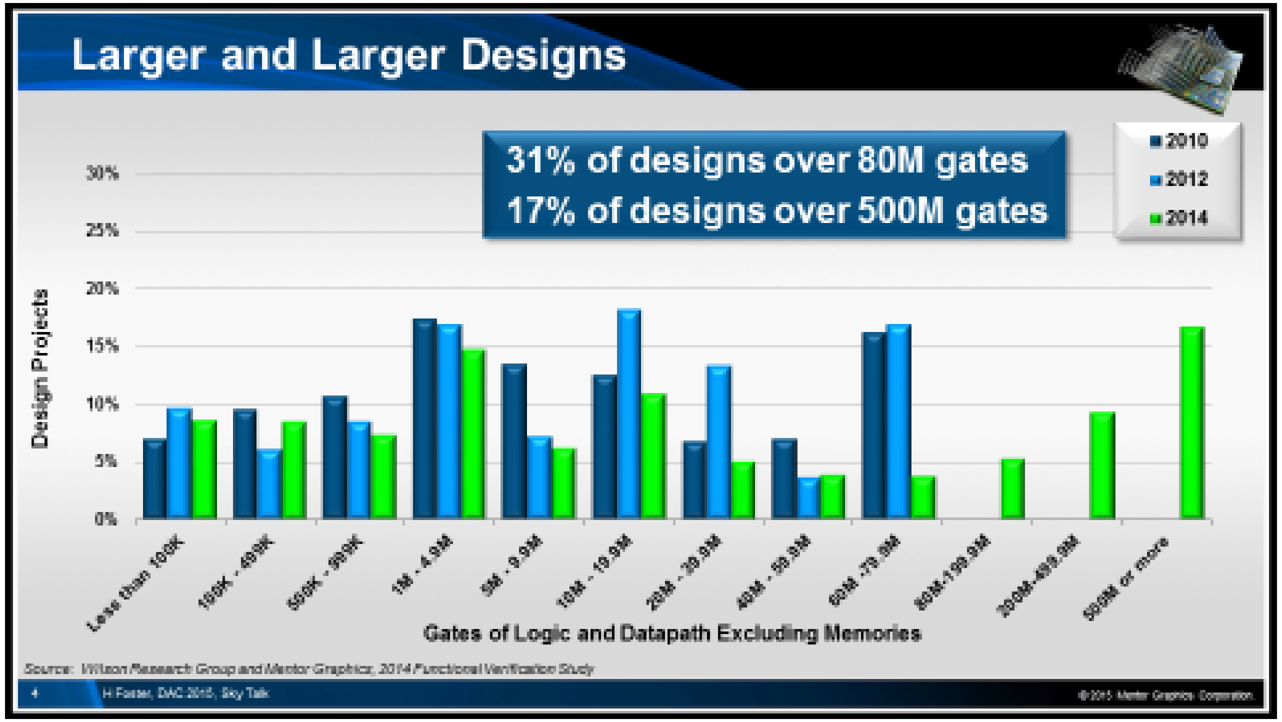
Source: arm.com

EDA Necessity

- Complexity



Source: intel.com



Larger and Larger Designs

31% of designs over 80M gates
17% of designs over 500M gates

Source: Wilson Research Group and Mentor Graphics, 2014 Functional Verification Study

4 H Foster, DAC 2015, Sky Talk

©2015 Mentor Graphics Corporation

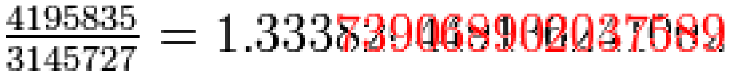
Source: Harry Foster, Mentor Graphics

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

Hardware Verification Necessity

High cost of failure

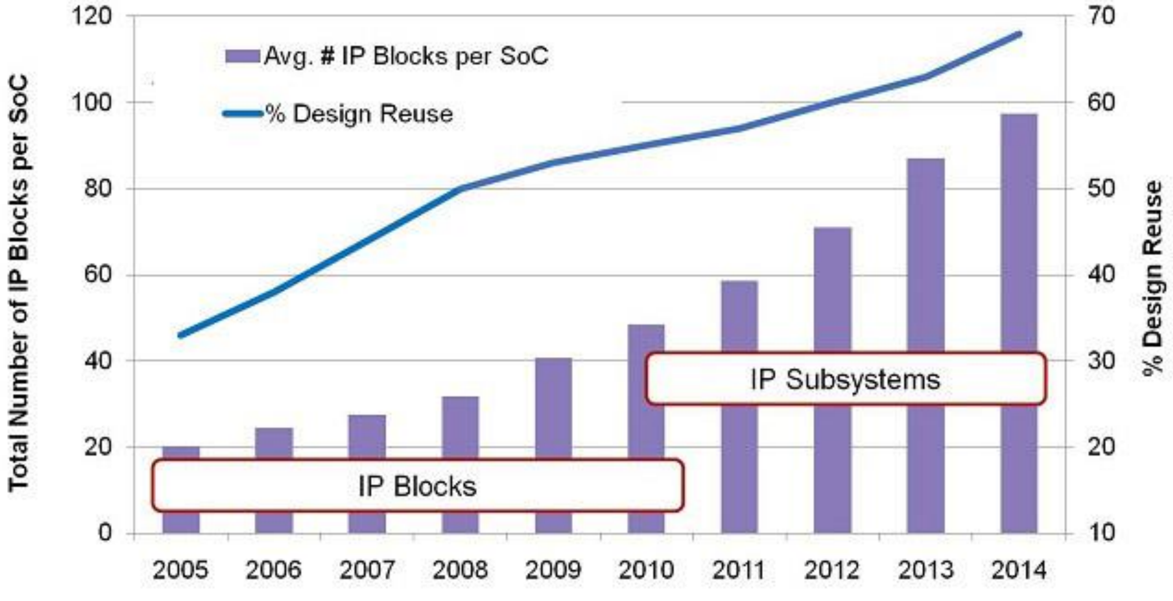
- Need for first silicon success
  - High mask costs
- Product recalls
  - Intel Pentium FDIV Bug 1994
  - Total cost: $475 million



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

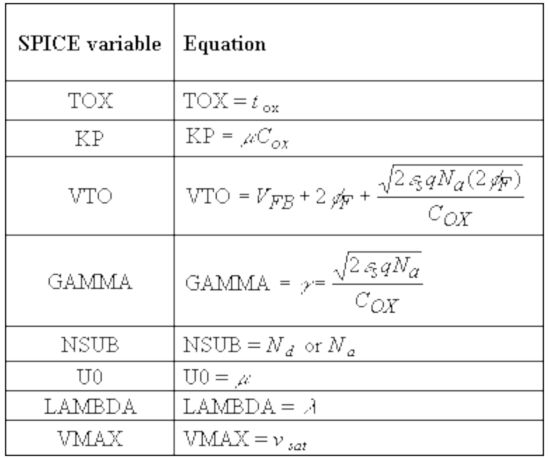


Source: Semico, October 2010

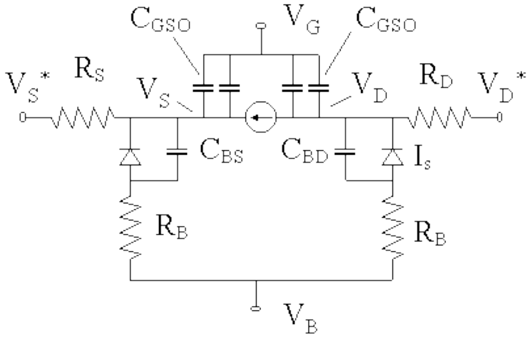
Benefits from scalability, diversity, cost reduction, novel functionality compelling enough for NDA?

EDA Evolution

- Models
  - Spice
    - Equations to model semiconductor devices



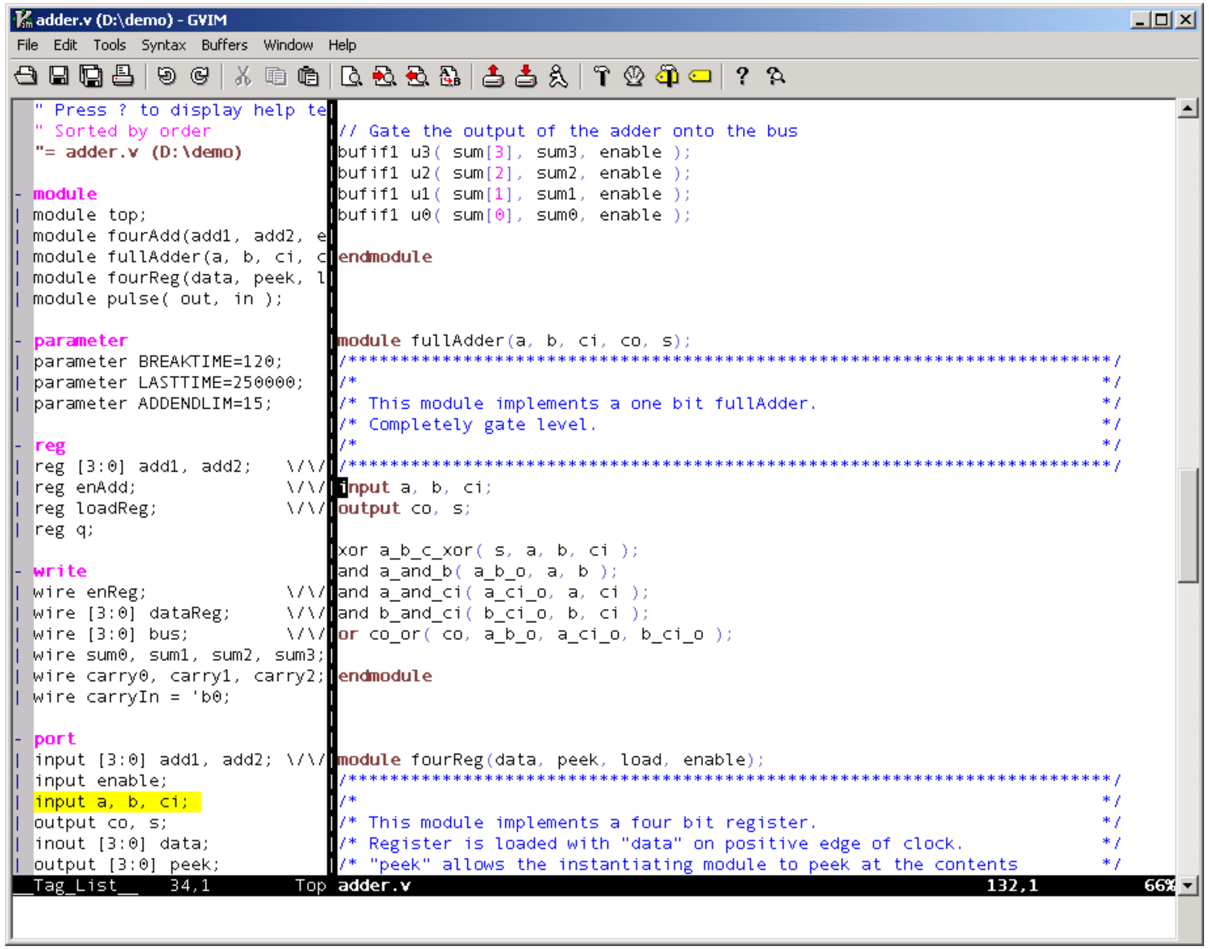
SPICE variable	Equation
TOX	$\mathrm{TOX} = t_{ox}$
KP	$\mathrm{KP} = \mu C_{ox}$
VTO	$\mathrm{VTO} = V_{FB} + 2\phi_F + \dfrac{\sqrt{2\varepsilon_s q N_a (2\phi_F)}}{C_{OX}}$
GAMMA	$\mathrm{GAMMA} = \gamma = \dfrac{\sqrt{2\varepsilon_s q N_a}}{C_{OX}}$
NSUB	$\mathrm{NSUB} = N_d$ or N_a
U0	$\mathrm{U0}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



Source: ecee.colorado.edu

EDA Evolution

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



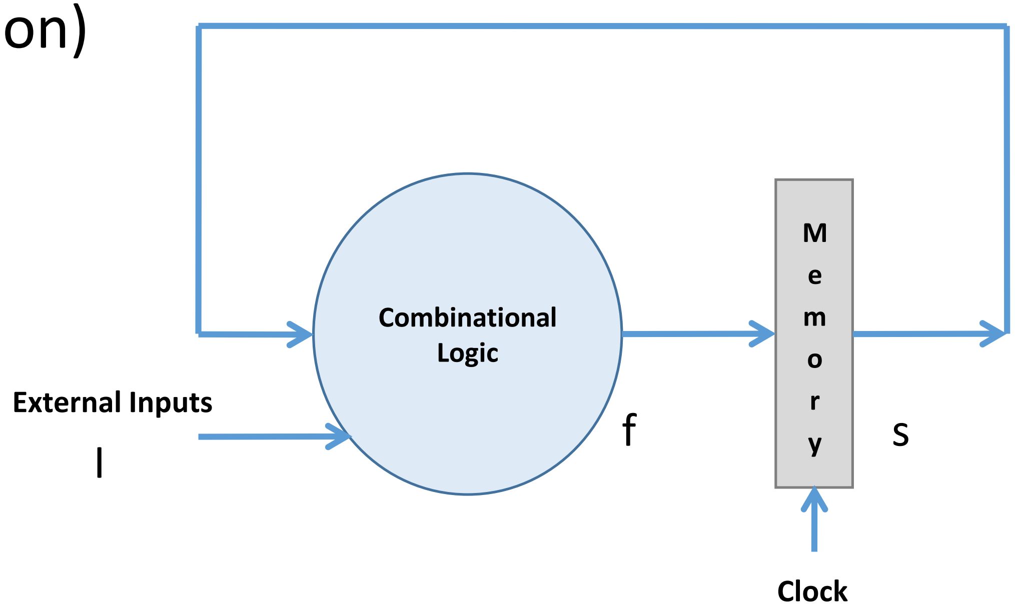
```
" Press ? to display help te
" Sorted by order
"= adder.v (D:\demo)

- 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;
__Tag_List__    34,1        Top
```

```

// 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



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



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        */
adder.v                                                     132,1     66%
```

Source: vim-taglist.sourceforge.net

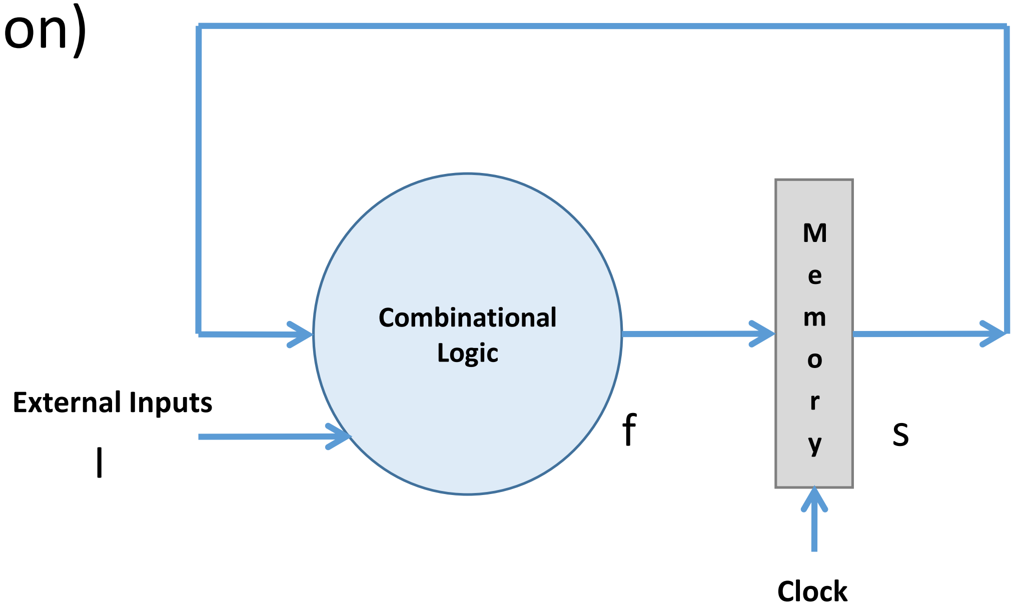
EDA Evolution

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



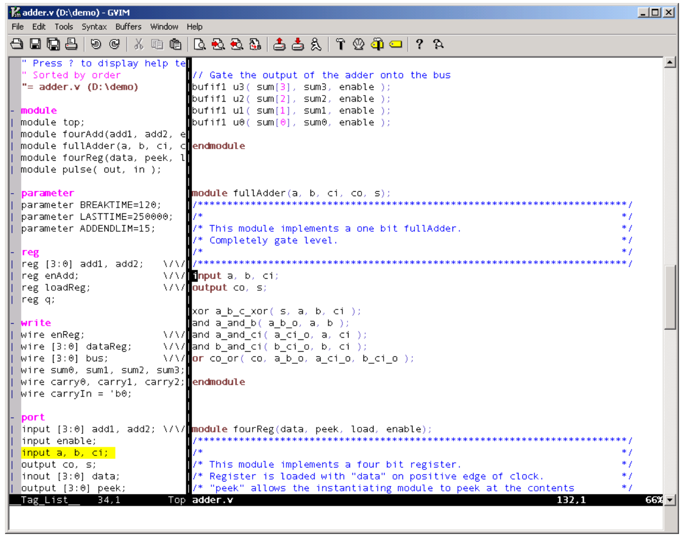
EDA Evolution

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



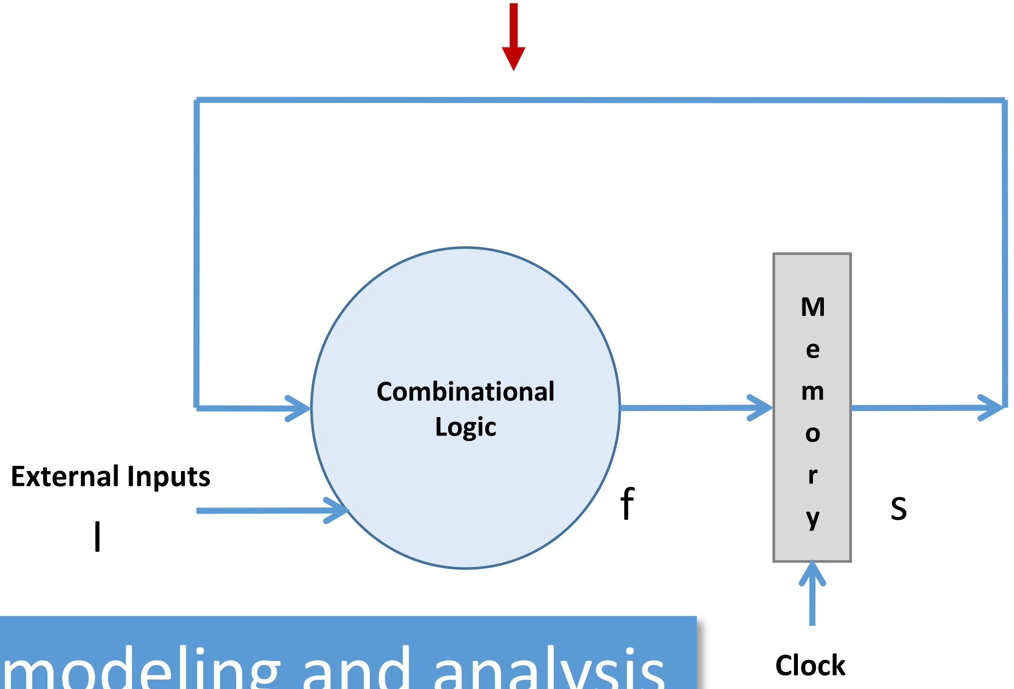
EDA Evolution

- 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



```
" 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;
__Tag_List__   34,1          Top
```

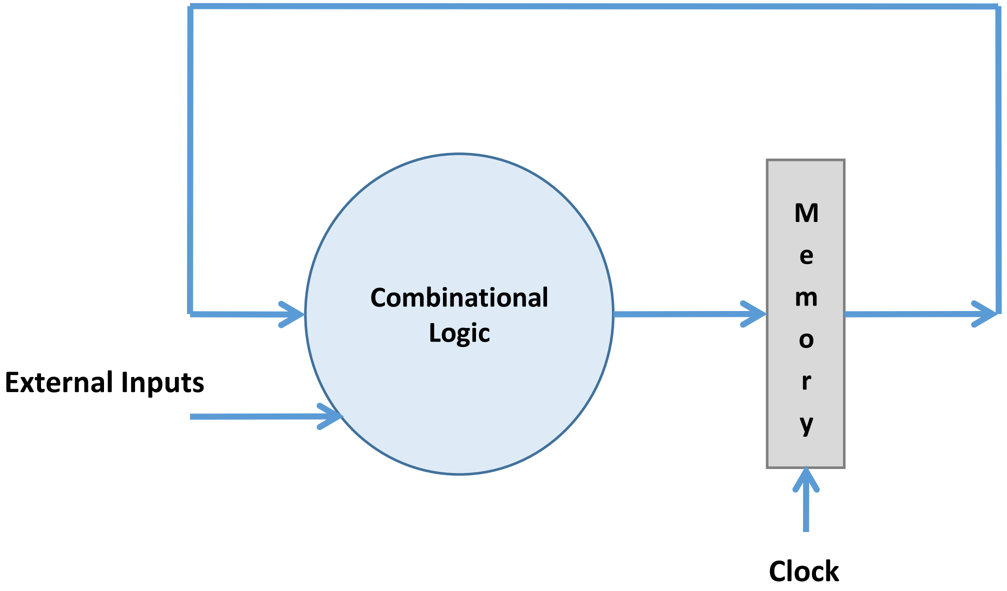
```

// 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



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



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          */
adder.v                                                    132,1       66%
```



Effective modeling and analysis for enabling NDA?

Analysis Capability Impacts Design Methodology

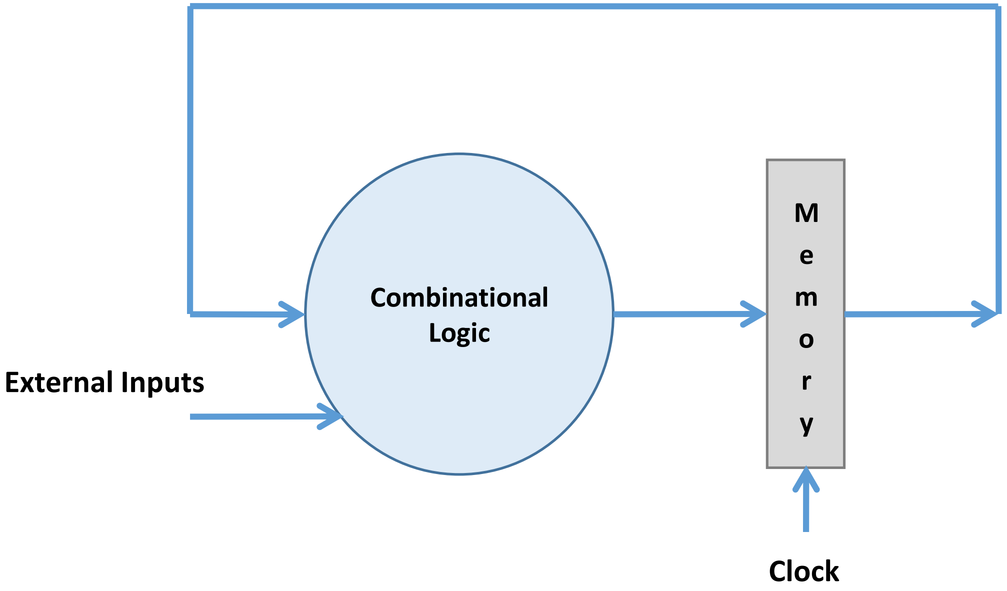
- Separation of Concerns
  - Separating timing and functional verification
    - Static timing analysis ignores functionality
    - Functional verification ignores timing
  - Driver for synchronous design methodology



Maximizing separation of concerns in Network Design?

Analysis Capability Impacts Design Methodology

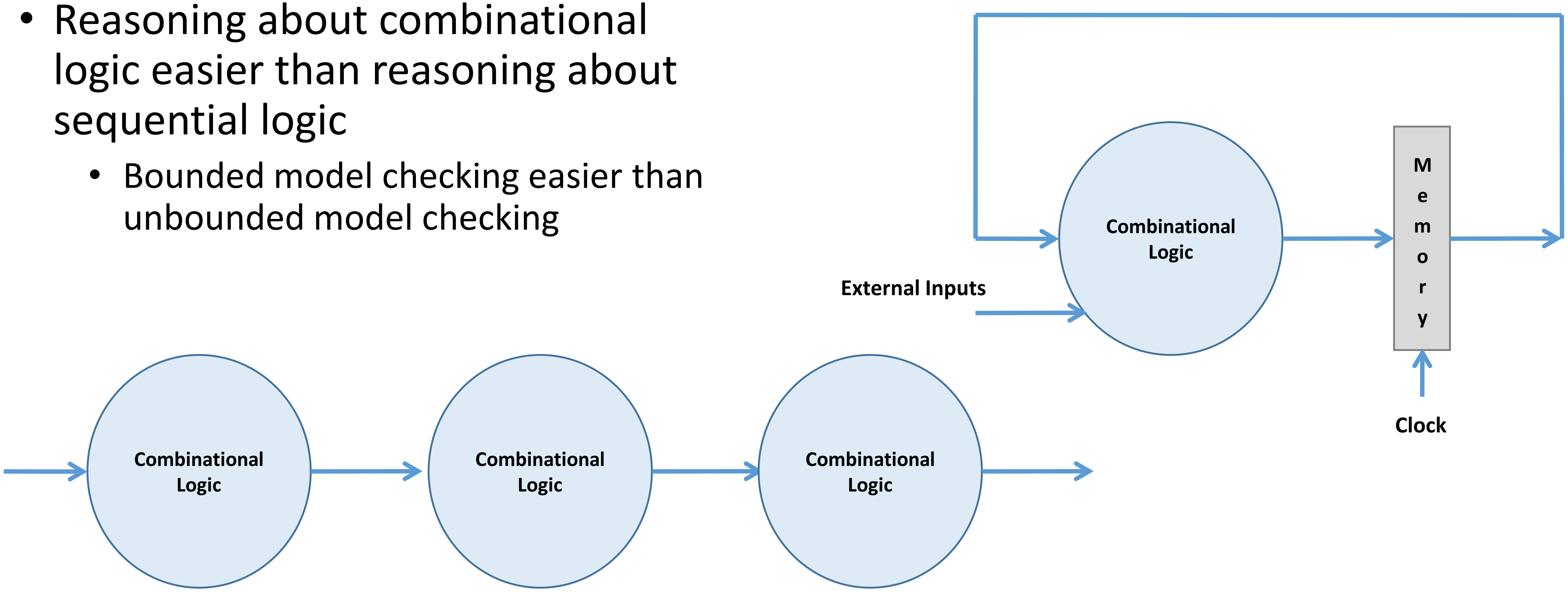
- Design Verification
  - Reasoning about combinational logic easier than reasoning about sequential logic
    - Designs obey initial register placements
      - state definition
    - Implementation verification reduces to combinational equivalence checking



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

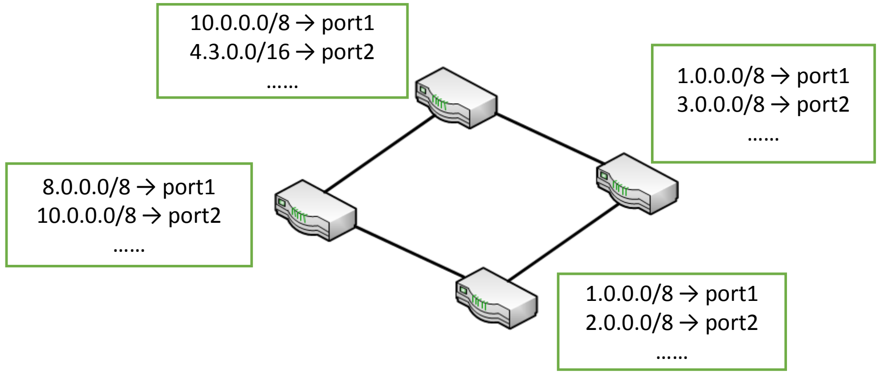
- Large State Space
  - Packet Header Size ≈ 100s of bits
    - MAC address: 48 bits
    - IP address: 32 bits
    - TCP port: 16 bits
    - VLAN, In port, Ethernet type…
  - Network Size
    - Tens to thousands of switches
    - Each switch generally has 1k~5k rules
    - Buffers…
  - Concurrency
    - 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



10.0.0.0/8 → port1
4.3.0.0/16 → port2
……

1.0.0.0/8 → port1
3.0.0.0/8 → port2
……

8.0.0.0/8 → port1
10.0.0.0/8 → port2
……

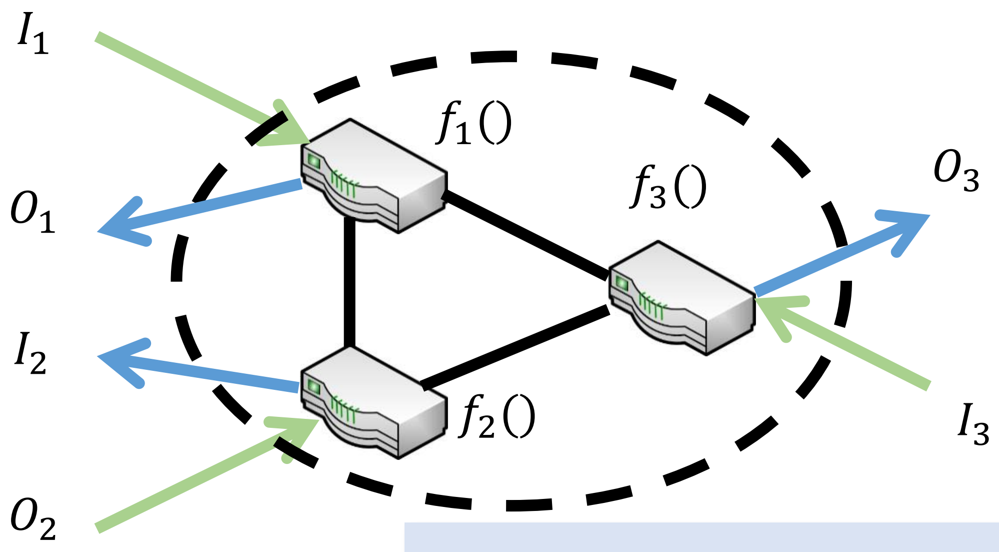
1.0.0.0/8 → port1
2.0.0.0/8 → port2
……

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

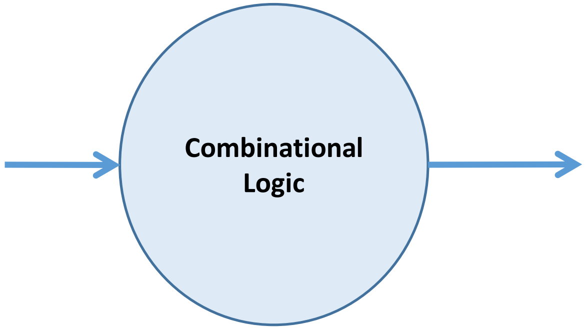
[1] Gude, N., Koponen, T., Pettit, J., Pfa, B., Casado, M., McKeown, N., Shenker, S.: "Nox: towards an operating system for networks," SIGCOMM 2008

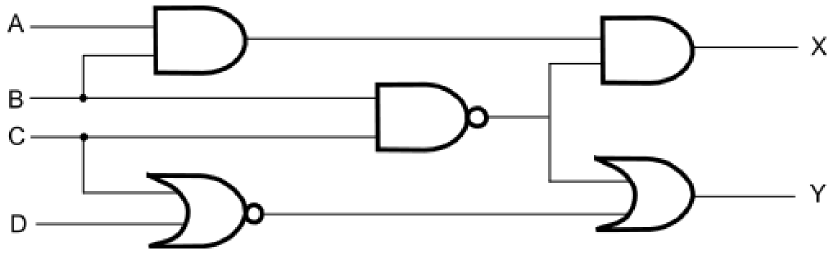
Data Plane as a Logic Circuit



Network Formula:

$N(I,O) = f_1 \wedge f_2 \wedge f_3$

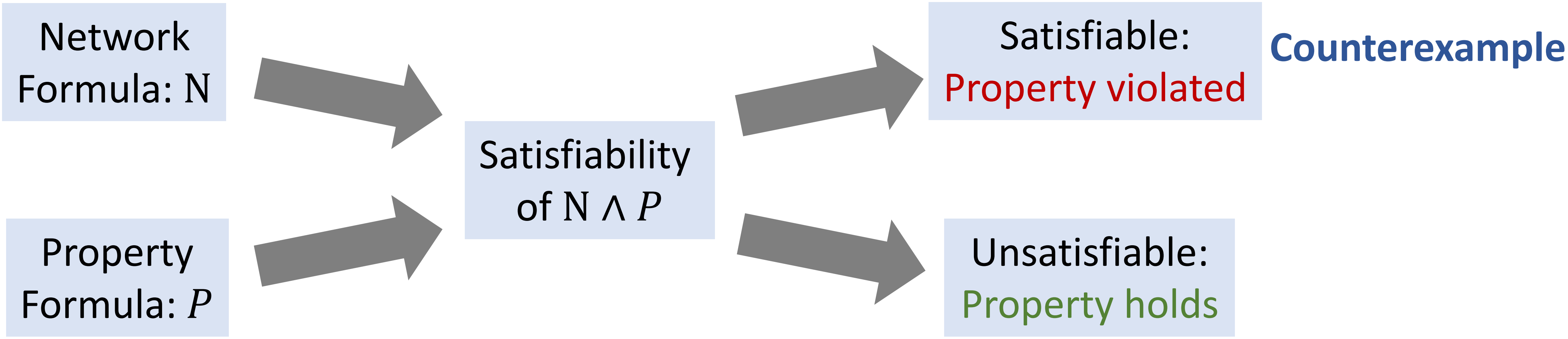




- Model it as a combinational logic circuit?
  - Outputs and signals are functions of only the present value of the inputs

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



Network Formula: N

Property Formula: P

Satisfiability of N $\wedge$ P

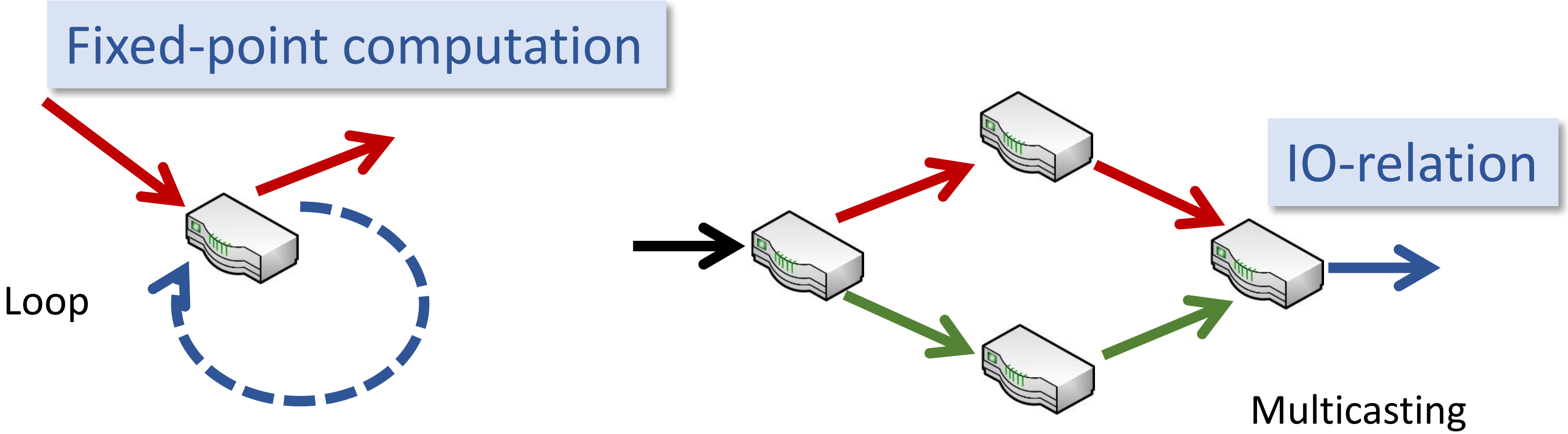
Satisfiable: Property violated

Counterexample

Unsatisfiable: Property holds

Modeling/Analysis Challenge

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



Fixed-point computation

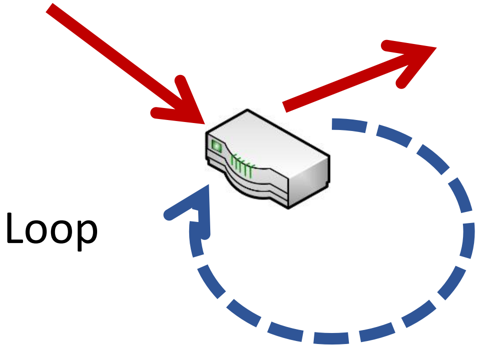
IO-relation

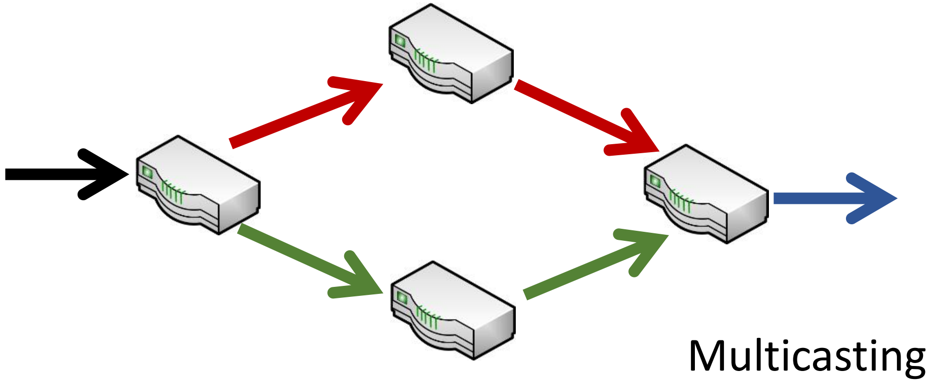
- Switch output not a combinational function of its inputs

Need to store sets of values

Adapting Modeling/Analysis

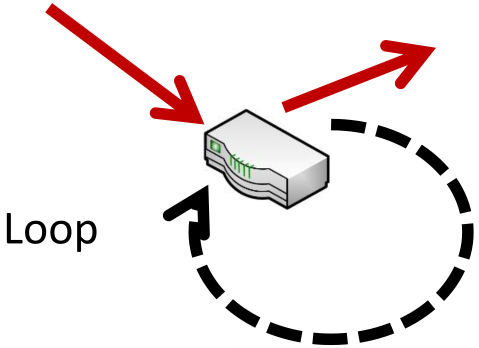
- Limit packet flow to a *single path for a single packet* through the network



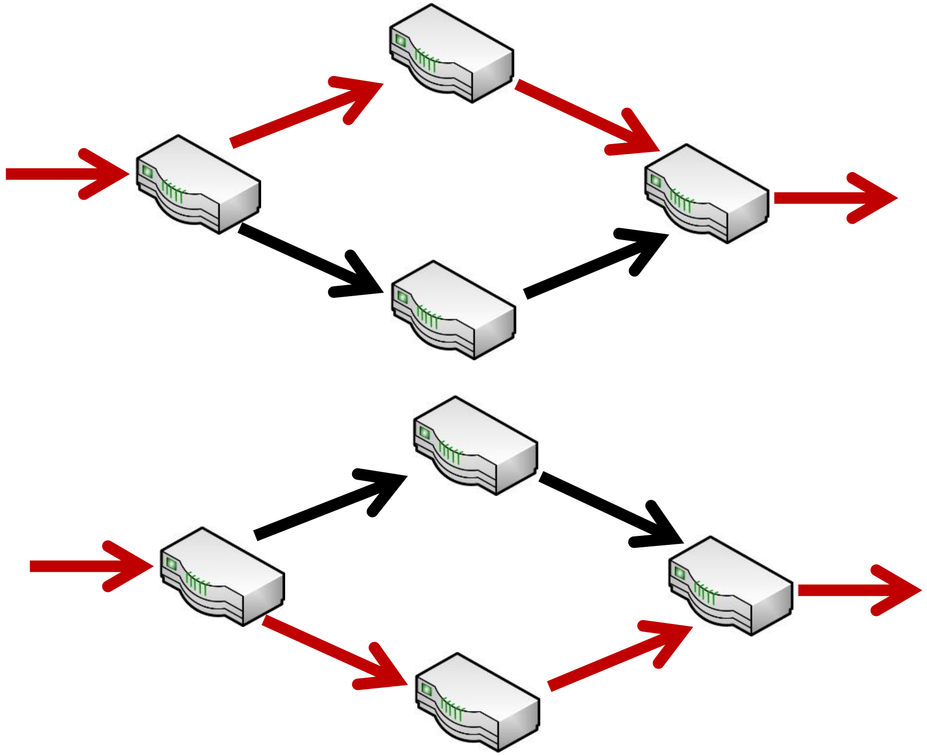


Adapting Modeling/Analysis

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



Loops implicitly blocked



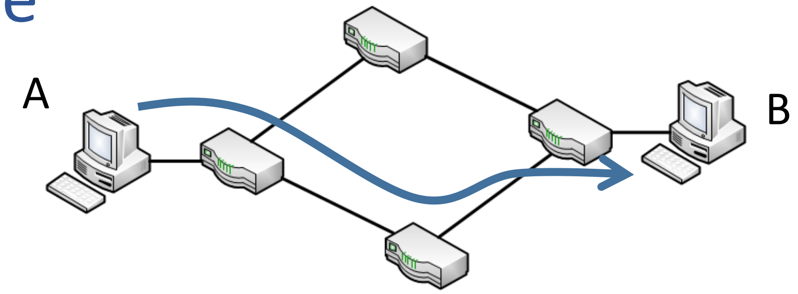
- Captures only part of the network behavior
- What good is this?

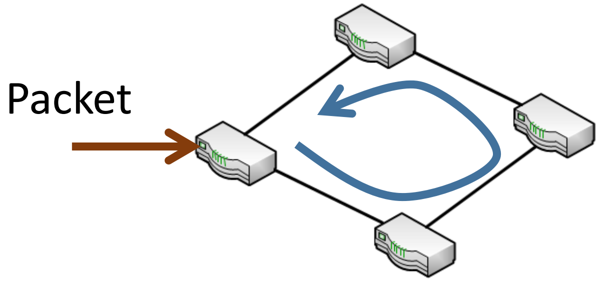
Goal: Counterexamples for Property Failures

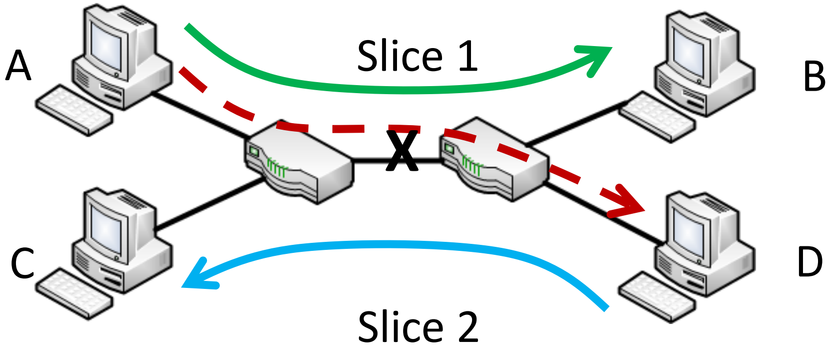
Single Path Single Packet Counterexample

Suffices for

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

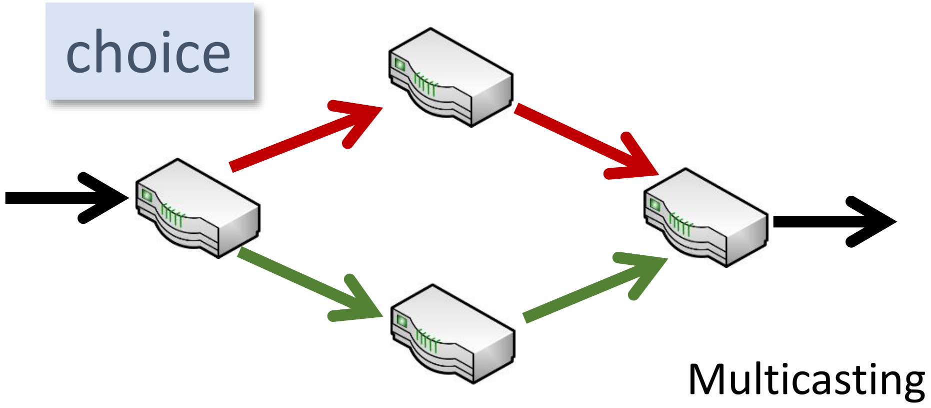






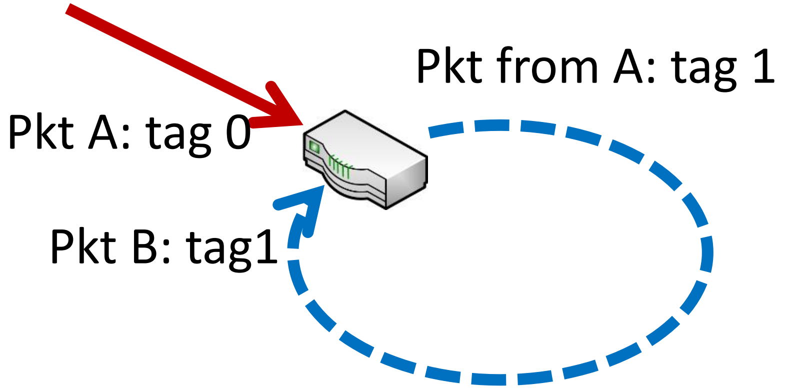
Adapting Modeling/Analysis

- 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

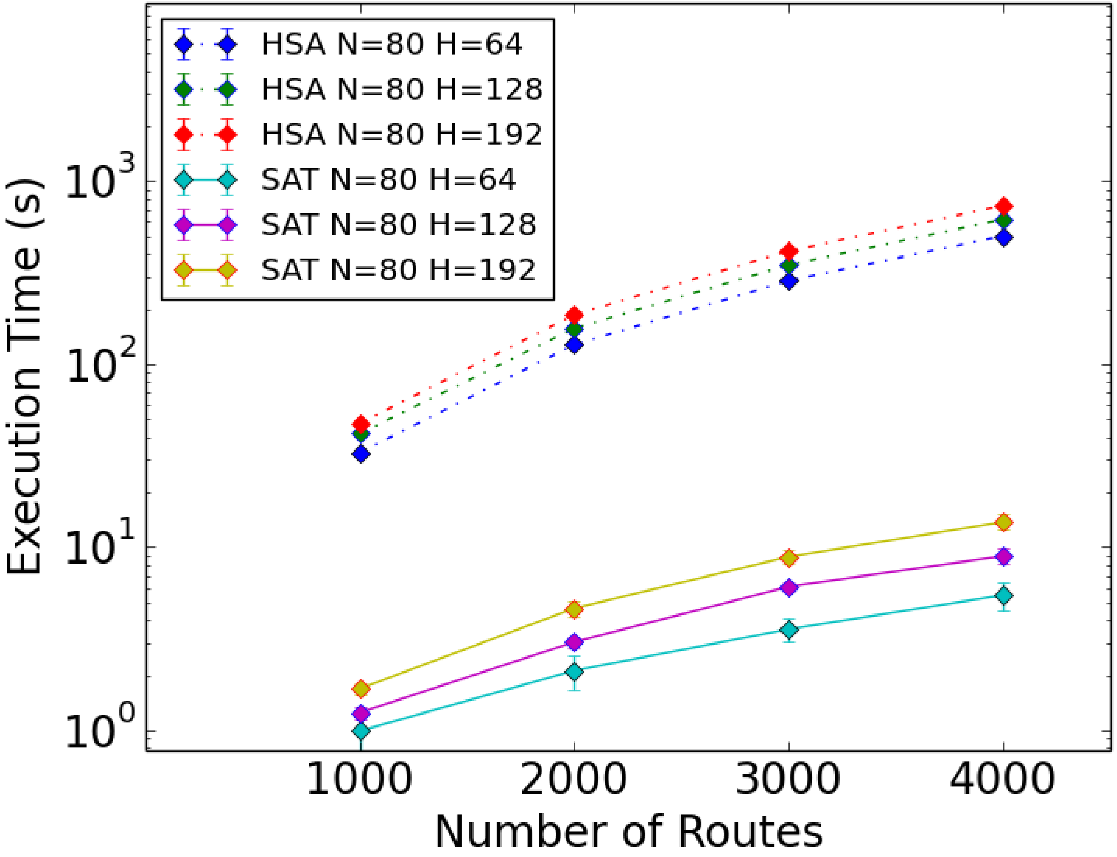
Experimental Results

Setup

- SAT solver: Minisat
- Stanford backbone network
  - 16 routers with full network functions (VLAN, ACL, …)
  - ≈ 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

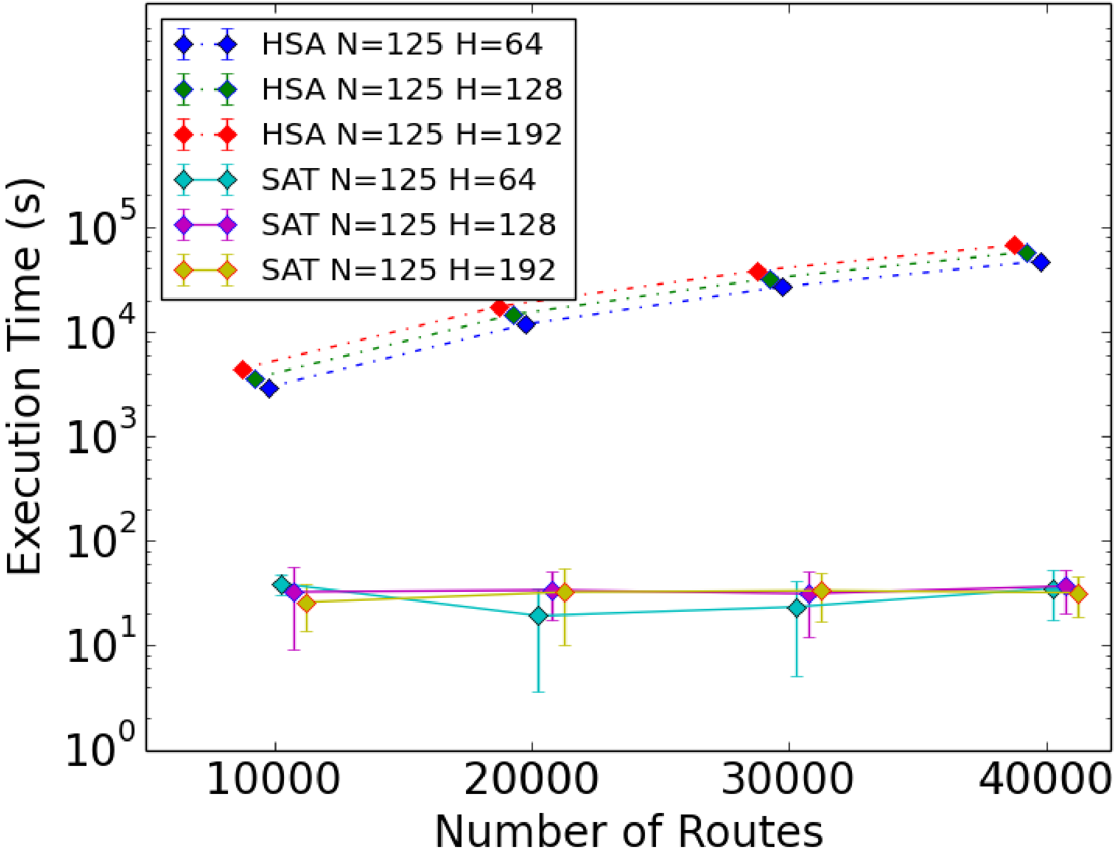
Experimental Results

- 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



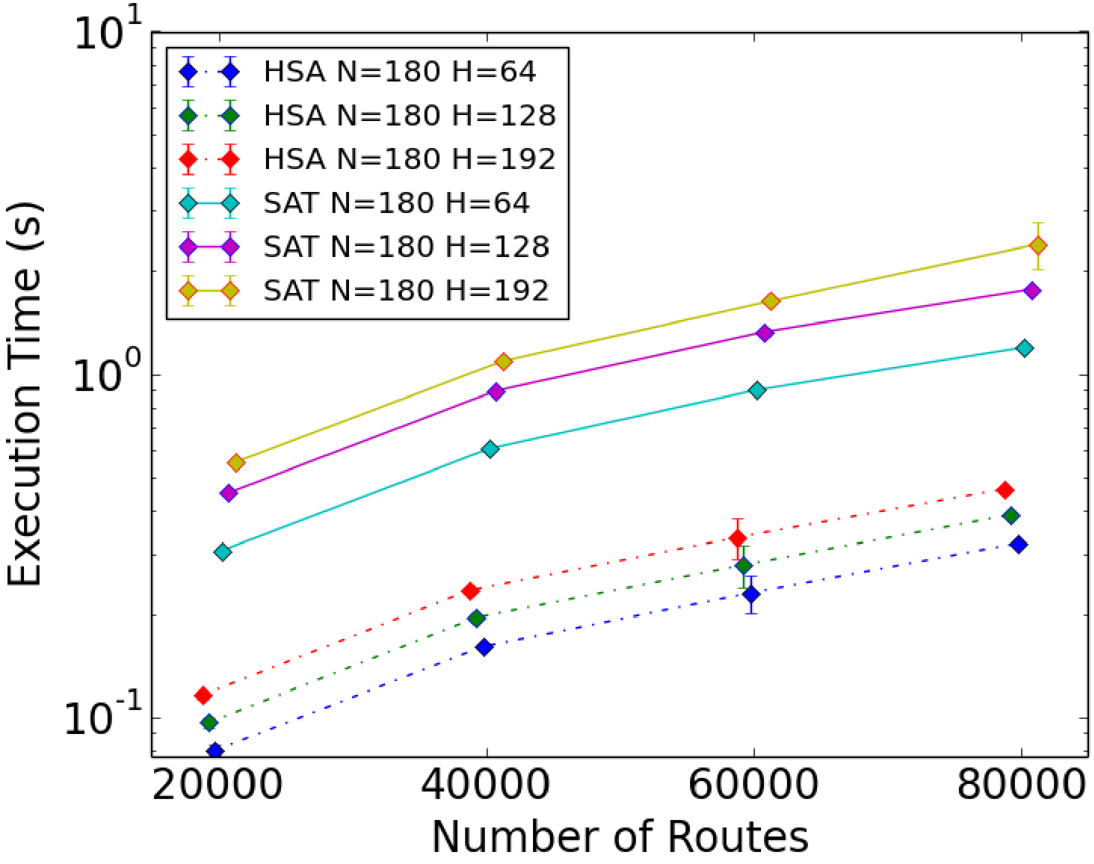
Experimental Results

- 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



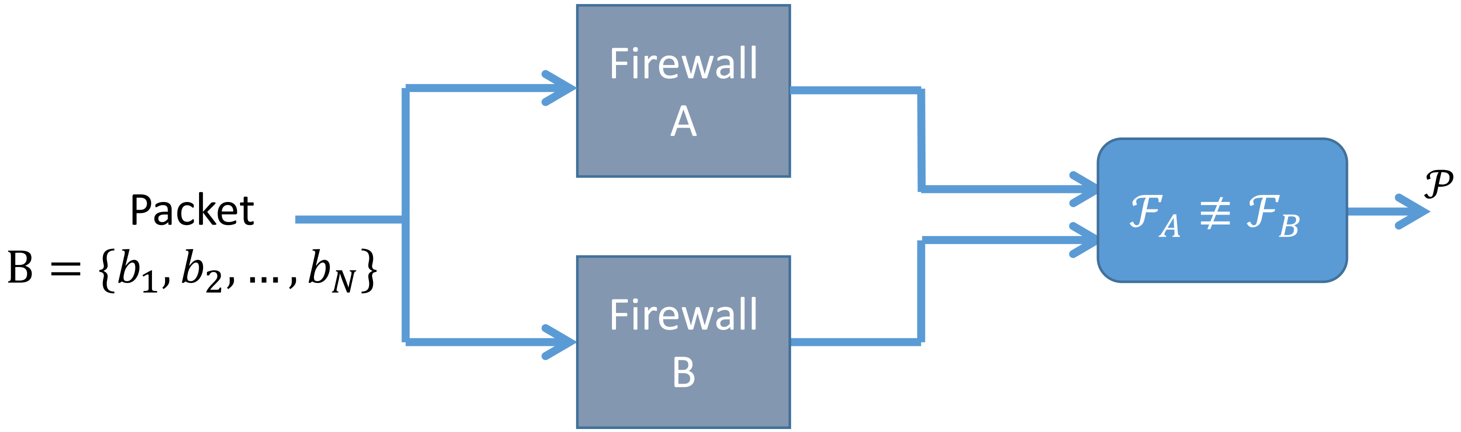
Experimental Results

- 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



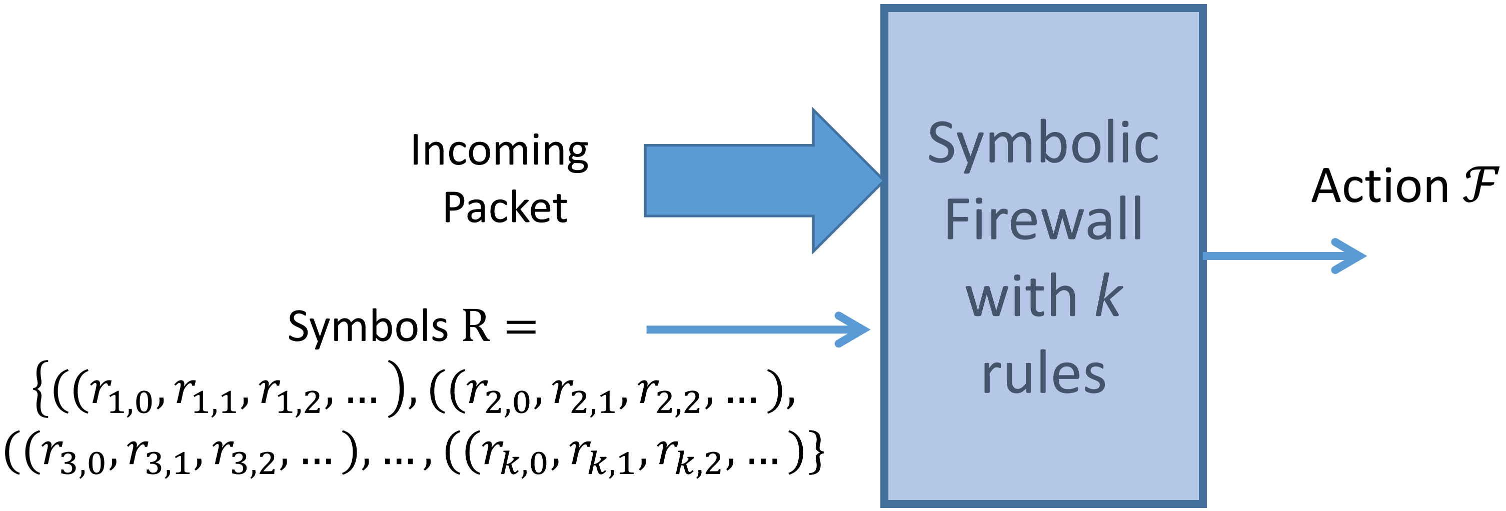
From Analysis to Synthesis: Firewall Case Study

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


Packet
$\mathrm{B} = \{b_1, b_2, \ldots, b_N\}$
Firewall A
Firewall B
$\mathcal{F}_A \not\equiv \mathcal{F}_B$
$\mathcal{P}$

Firewall Synthesis

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



Incoming Packet

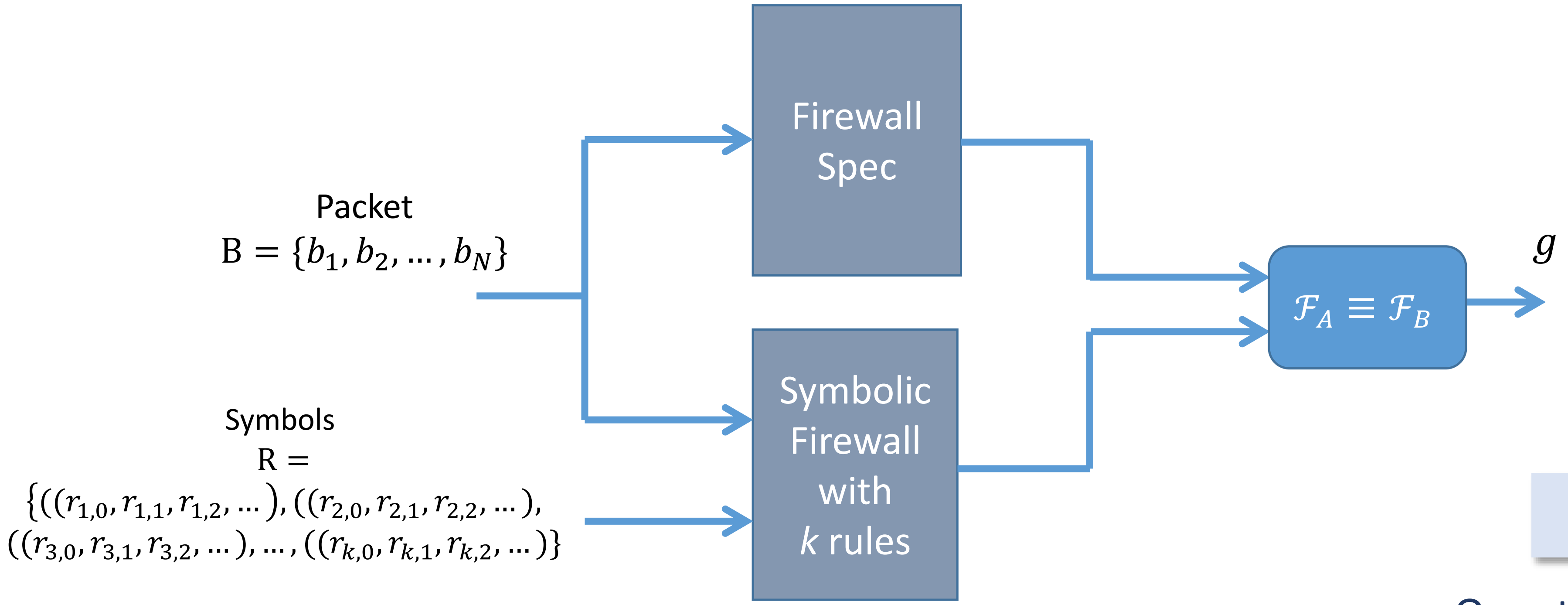
Symbols $\mathrm{R} =$
$\{((r_{1,0}, r_{1,1}, r_{1,2}, \ldots), ((r_{2,0}, r_{2,1}, r_{2,2}, \ldots),$
$((r_{3,0}, r_{3,1}, r_{3,2}, \ldots), \ldots, ((r_{k,0}, r_{k,1}, r_{k,2}, \ldots)\}$

Symbolic Firewall with *k* rules

Action $\mathcal{F}$

- Each assignment to R specifies one firewall

Firewall Synthesis



Packet

$\mathrm{B} = \{b_1, b_2, \dots, b_N\}$

Symbols

$\mathrm{R} =$

$\{((r_{1,0}, r_{1,1}, r_{1,2}, \dots), ((r_{2,0}, r_{2,1}, r_{2,2}, \dots),$

$((r_{3,0}, r_{3,1}, r_{3,2}, \dots), \dots, ((r_{k,0}, r_{k,1}, r_{k,2}, \dots)\}$

Firewall Spec

Symbolic Firewall with *k* rules

$\mathcal{F}_A \equiv \mathcal{F}_B$

g

$$\exists R \; \forall B \;\; (g)$$

Quantified Boolean Formula (QBF)

- 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?