Formally Verifiable Networking

Anduo Wang\textsuperscript{1} Limin Jia\textsuperscript{1} Changbin Liu\textsuperscript{1}
Boon Thau Loo\textsuperscript{1} Andre Scedrov\textsuperscript{1}
Oleg Sokolsky\textsuperscript{1} Prithwish Basu\textsuperscript{2}

\textsuperscript{1}University of Pennsylvania
\textsuperscript{2}BBN technologies

Protocol eXchange Oct 7, 2009
Outline

Motivation

Overview
  Background on Declarative Networking

Verification in FVN
  NDLog Program Verification
  Verified Code Generation

Meta-Theoretic Model
  Compositional Routing Algebra

Conclusion

Open issues
Motivation

Overview
  Background on Declarative Networking

Verification in FVN
  NDLog Program Verification
  Verified Code Generation

Meta-Theoretic Model
  Compositional Routing Algebra

Conclusion

Open issues
Motivation

▶ Challenges to today’s Internet
  ▶ Unwanted and harmful traffic
  ▶ Complexity and fragility in Internet routing
Motivation

- Challenges to today’s Internet
  - Unwanted and harmful traffic
  - Complexity and fragility in Internet routing
- New applications demand new capabilities
  - Resiliency (RON, SOSR, Detour...)
  - Scalable Lookup (Chord, Pastry, Tapestry,...)
  - Mobility (i3, DHARMA, HIP)
  - Security (SOS, OverDoSe)
  - Content-distribution (Akamai, CoralCDN)
  - Multicast (Overcast, ESM)
Motivation

- Challenges to today’s Internet
  - Unwanted and harmful traffic
  - Complexity and fragility in Internet routing
- New applications demand new capabilities
  - Resiliency (RON, SOSR, Detour…)
  - Scalable Lookup (Chord, Pastry, Tapestry,…)
  - Mobility (i3, DHARMA, HIP)
  - Security (SOS, OverDoSe)
  - Content-distribution (Akamai, CoralCDN)
  - Multicast (Overcast, ESM)
- Clean-slate Internet Design
  - NSF FIND (Future Internet Design)
  - GENI (Global Environment for Network Innovation)
Motivation

- Challenges to today’s Internet
  - Unwanted and harmful traffic
  - Complexity and fragility in Internet routing
- New applications demand new capabilities
  - Resiliency (RON, SOSR, Detour…)
  - Scalable Lookup (Chord, Pastry, Tapestry,…)
  - Mobility (i3, DHARMA, HIP)
  - Security (SOS, OverDoSe)
  - Content-distribution (Akamai, CoralCDN)
  - Multicast (Overcast, ESM)
- Clean-slate Internet Design
  - NSF FIND (Future Internet Design)
  - GENI (Global Environment for Network Innovation)

Needed: Better software tools for deploying and analyzing new network protocols and architectures
Recent Efforts in Network Design and Verification

- Correct-by-construction Meta-model
  - Metarouting [SIGCOMM’05]
- Runtime verification
  - Pip [NSDI’06]
  - DS3 [NSDI’08]
- Model checking
  - MaceMC [NSDI’07] best paper
  - CMC [NSDI’04]
Limitations of Current Approaches

- Metarouting
  - Idealized model unlikely to be adapted to actual implementation
Limitations of Current Approaches

- **Metarouting**
  - Idealized model unlikely to be adapted to actual implementation

- **Runtime verification**
  - Incur additional runtime overhead
  - Non-exhaustive, limited class of properties

- **Model checking network implementation**
  - Require model extraction
  - State explosion problem.
Limitations of Current Approaches

- **Metarouting**
  - Idealized model unlikely to be adapted to actual implementation

- **Runtime verification**
  - Incur additional runtime overhead
  - Non-exhaustive, limited class of properties

- **Model checking network implementation**
  - Require model extraction
  - State explosion problem.

- **Classical theorem proving**
  - High initial investment in formal specification
  - Restricted to design and standard, decoupled from actual implementation
Motivation

Overview
  Background on Declarative Networking

Verification in FVN
  NDLog Program Verification
  Verified Code Generation

Meta-Theoretic Model
  Compositional Routing Algebra

Conclusion

Open issues
Formally Verifiable Networking
unifying the design, specification, implementation, and
verification of networking protocols

- *Formal logical statements* specify the behavior and the
  properties of the protocol
- *Theorem proving* establishes correctness of formal system
  specification w.r.t network properties
Formally Verifiable Networking
unifying the design, specification, implementation, and verification of networking protocols

- *Formal logical statements* specify the behavior and the properties of the protocol
- *Theorem proving* establishes correctness of formal system specification w.r.t network properties
- *Declarative networking*, intermediary layer between logical specification and real implementation
  - Property preserving translations from declarative networking implementation to formal system specifications for verification
  - Code generation from verified formal specification
Formally Verifiable Networking
unifying the design, specification, implementation, and verification of networking protocols

- Formal logical statements specify the behavior and the properties of the protocol
- Theorem proving establishes correctness of formal system specification w.r.t network properties
- Declarative networking, intermediary layer between logical specification and real implementation
  - Property preserving translations from declarative networking implementation to formal system specifications for verification
  - Code generation from verified formal specification
- Meta-model, correctness-by-construction design
**Design**: correctness-by-construction via meta-model
Design: correctness-by-construction via meta-model

Specification: two way property preserving translation
  ▶ Formal system specification generated from NDlog program (arc 4)
  ▶ Executable Declarative network synthesized from verified logical specification (arc 3)
Design: correctness-by-construction via meta-model
Speciﬁcation: two way property preserving translation
  ▶ Formal system speciﬁcation generated from NDlog program (arc 4)
  ▶ Executable Declarative network synthesized from veriﬁed logical speciﬁcation (arc 3)
Verification: proving network invariants of system speciﬁcations by interacting with theorem prover (arc 5)
FVN Overview

- **Design**: correctness-by-construction via meta-model
- **Specification**: two way property preserving translation
  - Formal system specification generated from NDlog program (arc 4)
  - Executable Declarative network synthesized from verified logical specification (arc 3)
- **Verification**: proving network invariants of system specifications by interacting with theorem prover (arc 5)
- **Implementation**: distributed query processing (arc 7)
Motivation

Overview

Background on Declarative Networking

Verification in FVN

NDLog Program Verification
Verified Code Generation

Meta-Theoretic Model

Compositional Routing Algebra

Conclusion

Open issues
Declarative specifications of networks using *Network Datalog* (NDLog), a distributed variant of Datalog.

NDLog is compiled to distributed dataflows (arc 7).

Distributed query processor executes the dataflows to implement the network protocols.
Ease of programming:
- Compact high-level representation of protocols
- Orders of magnitude reduction in code size

Ease of analysis:
- Amenable to static analysis and theorem proving

See Loo et. al [SOSP ’05, SIGMOD ’06] for implementation details of declarative networking
Network Datalog (NDlog) by example
All-Pairs Reachability

R1: reachable(@S,D)<-link(@S,D)
R2: reachable(@S,D)<-link(@S,Z),reachable(@Z,D)

- input: link(@S,D), output: reachable(@S,D)
- link(@S,D): a link from node S to D, reachable(@S,D): node S can reach D
- Location specifier: value of attribute prefixed with @ determines the location of each tuple
Network Datalog (NDlog) by example
All-Pairs Reachability

▷ R1: reachable(@S,D)<-link(@S,D)
R2: reachable(@S,D)<-link(@S,Z),reachable(@Z,D)

▷ For all nodes S,D: S can reach D if there is a link from S to D

▷ input: link(@S,D), output: reachable(@S,D)
▷ link(@S,D): a link from node S to D, reachable(@S,D): node S can reach D
▷ Location specifier: value of attribute prefixed with @ determines the location of each tuple
Network Datalog (NDlog) by example

All-Pairs Reachability

R1: \texttt{reachable}(@S,D) \leftarrow \texttt{link}(@S,D)

R2: \texttt{reachable}(@S,D) \leftarrow \texttt{link}(@S,Z),\texttt{reachable}(@Z,D)

- For all nodes $S,D,Z$: if there is a link from $S$ to $Z$, and that $Z$ can reach $D$, then $S$ can reach $D$

- **input:** \texttt{link}(@S,D), **output:** \texttt{reachable}(@S,D)

  - \texttt{link}(@S,D): a link from node $S$ to $D$, \texttt{reachable}(@S,D): node $S$ can reach $D$

  - **Location specifier:** value of attribute prefixed with @ determines the location of each tuple
Declarative Networking in Practice

- Example implementations to date:
  - Wired and wireless routing protocols (DV, LS, DSR, AODV, OLSR, etc.) \[\text{[SIGCOMM’05, ICNP’09]}\]
  - Chord Distributed Hash Table \[\text{[SOSP’05]}\]
  - Resilient overlay network (RON) \[\text{[CoNEXT’08]}\]
  - Internet Indirection Infrastructure (i3) \[\text{[CoNEXT’08]}\]
  - Others: sensor networking protocols \[\text{[Sensys’07]}\], multicast overlays, replication, snapshot, fault tolerance

- P2 declarative networking system
  - http://p2.cs.berkeley.edu

- A declarative toolkit for rapid network protocol simulation and experimentation \[\text{[SIGCOMM’09 demo]}\]
  - http://netdb.cis.upenn.edu/rapidnet/
Motivation

Overview
  Background on Declarative Networking

Verification in FVN
  NDLog Program Verification
  Verified Code Generation

Meta-Theoretic Model
  Compositional Routing Algebra

Conclusion

Open issues
Automatic property preserving translations (arc 4): from declarative networking implementation to formal system specifications recognizable by existing theorem provers

Verifying network properties (arc 5) by proving invariants against formal system specifications using theorem prover

Automatic NDlog code generation (arc 3) from verified component specification
Motivation

Overview
  Background on Declarative Networking

**Verification in FVN**
  NDLog Program Verification
  Verified Code Generation

Meta-Theoretic Model
  Compositional Routing Algebra

Conclusion

Open issues
Example NDlog Program Verification

- Path-Vector Protocol in NDlog program
  - p1  \( \text{path}(\text{@S,D,P,C}):- \text{link}(\text{@S,D,C}), P=(\text{S,D}). \)
  - p2  \( \text{path}(\text{@S,D,P,C}):- \text{link}(\text{@S,Z,C1}), \)
        \( \text{path}(\text{@Z,D,P2,C2}), C=C1+C2, P=\text{concatPath(Z,P2)}. \)

- Encoding path-vector in theorem prover
  - p1: \( \forall (S, D, P, C). \text{link}(S, D, C) \land P = f_{\text{init}}(S, D) \implies \text{path}(S, D, P, C) \)
  - p2: \( \forall (S, D, P, C). \exists (C_1, C_2, Z, P_2). \text{link}(S, Z, C_1) \land \)
        \( \text{bestPath}(Z, D, P_2, C_2) \land C = C_1 + C_2 \land P = \)
        \( f_{\text{concatPath}}(Z, P_2) \implies \text{path}(S, D, P, C) \)

- Proving Route optimality property in PVS
  - FORALL (S,D:Node) (C:Metric) (P:Path): \( \text{bestPath}(S,D,P,C) \implies \neg (\exists \{C_2:Metric\} \{P_2:Path\}: \)
        \( \text{path}(S,D,P_2,C_2) \land C_2 < C) \)
  - ("" (skosimp*) (expand bestPath) (prop) (expand bestPathCost) (prop) (skosimp*) (inst -2 C2!1) (grind))
Handling soft-state in networks

- Soft-state: network state expires after Time-To-Live (TTL) unless refreshed
- Ensures eventual consistency in protocol in the presence of message reordering and/or losses
- Additional rewrite step required for rules that uses soft-state predicates
Distance Vector Routing Protocol with Soft-State

- Distance vector routing:
  - NDLog specifications similar to path-vector routing except only next hop (instead of entire path) is traversed
- An instance of a soft-state NDLog program
  - Nodes periodically advertise to their neighbors their best known distances to other destinations
  - Nodes use these advertisements to select the best neighbor along the shortest path to destination
  - Advertisements timed-out unless refreshed
Example Properties Verified using Soft-State Distance Vector Protocol

- **Eventual convergence in stable network**
  
  $$\text{bestHopCost\_converge: THEOREM EXISTS } (j:\text{posnat}):$$
  \[
  \forall (S,D:\text{Node})(C:\text{Metric})(i:\text{posnat}): (i>j) \Rightarrow \text{bestHopCost}(S,D,C,5\times i,10) = \text{bestHopCost}(S,D,C,5\times j,10)
  \]

- **Divergence (count-to-infinity problem) in dynamic network**

- **A well known solution:** *split-horizon* can avoid count-to-infinity in two-node cycle, but cannot prevent the problem in three-node cycle
Motivation

Overview
   Background on Declarative Networking

Verification in FVN
   NDLog Program Verification
   Verified Code Generation

Meta-Theoretic Model
   Compositional Routing Algebra

Conclusion

Open issues
Component Based Verification of BGP System

AS W sends route update to AS U

AS U recomputes the best route R0' and exports to neighbors at the next time iteration

Specification of BGP components in PVS

- \text{bgp}(U,W,R0,R3,T): \text{INDUCTIVE bool} = \exists (R1,R2): \text{activeAS}(U,W,T) \land \text{pt}(U,W,R0,R3,T) \land \text{bestRoute}(W,T,R0)

- \text{pt}(U,W,R0,R3,T): \text{INDUCTIVE bool} = \text{export}(U,W,R0,R1,T) \land \text{pvt}(U,W,R1,R2,T) \land \text{import}(U,W,R2,R3,T)

More details on the PVS specifications and example proofs of various BGP systems

Generating Equivalent NDlog implementation

- Specify atomic component \( t \) as a rule taking \( t_{\text{in}}(I) \) as rule body, deriving \( t_{\text{out}}(O) \) as rule head
  - PVS specification \( t(I,O) : \text{INDUCTIVE bool} = CT(I,O) \)
  - The equivalent NDlog rule \( t_{\text{out}}(O) : - t_{\text{in}}(I), CT(I,O) \)

- Specify compositional component as set of rules

\[
\begin{align*}
tc(I1,I2,O3): & \text{INDUCTIVE bool} = \text{EXISTS} \\
& (O1,O2): t1(I1,O1) \text{ AND } t2(I2,O2) \text{ AND } t3(O1,O2,O3)
\end{align*}
\]

\[
\begin{align*}
t1(I,O): & \text{INDUCTIVE bool} = C1(I,O) \\
t2(I,O): & \text{INDUCTIVE bool} = C2(I,O) \\
t3(I,O',O): & \text{INDUCTIVE bool} = C3(I,I',O)
\end{align*}
\]

- Equivalent NDlog implementations
  - \( t1_{\text{out}}(O1) : - t1_{\text{in}}(I1), C1(I1,O1) \).
  - \( t2_{\text{out}}(O2) : - t2_{\text{in}}(I2), C2(I2,O2) \).
  - \( t3_{\text{out}}(O3) : - t1_{\text{out}}(O1), t2_{\text{out}}(O2), C3(O1,O2,O3) \).
Circuit-Like Component Representation of NDlog Program

\[ a1: \text{hop}(\text{@S}, \text{D}, \text{D}, \text{C}, \text{Tc}, 10) : - \text{link}(\text{@S}, \text{D}, \text{C}, \text{Tc}, 10). \]
\[ a2: \text{hop}(\text{@S}, \text{D}, \text{Z}, \text{C}, \text{Tc}, 10) : - \text{hopMsg}(\text{@S}, \text{D}, \text{Z}, \text{C}, \text{Tc}2), \text{Tc} = \text{Tc}2 + 5 \]
\[ \text{agg} \text{ bestHopCost}(\text{@S}, \text{D}, \text{min}<\text{C}>, \text{Tc}, 10) : - \text{hop}(\text{@S}, \text{D}, \text{D}, \text{C}, \text{Tc}, 10). \]
\[ b: \text{hopMsg}(\text{@N}, \text{D}, \text{Z}, \text{C}, \text{Tc}, 0) : - \text{periodic} \_ \text{dv}(\text{@S}, 5, \text{Tc}), \text{link}(\text{@S}, \text{N}, \text{C2}, \text{Tc2}, 10), \text{bestHop}(\text{@S}, \text{D}, \text{Z}, \text{C1}, \text{Tc1}, 10), \text{C} = \text{C1} + \text{C2}, \text{Tc2} < \text{Tc} \leq \text{Tc2} + 10, \text{Tc1} < \text{Tc} \leq \text{Tc1} + 10. \]
\[ c: \text{bestHop}(\text{@S}, \text{D}, \text{Z}, \text{C}, \text{Tc}, 10) : - \text{bestHopCost}(\text{@S}, \text{D}, \text{C}, \text{Tc}, 10), \text{hop}(\text{@S}, \text{D}, \text{Z}, \text{C}, \text{Tc1}, 10), \text{Tc1} < \text{Tc} \leq \text{Tc1} + 10. \]
Motivation

Overview
  Background on Declarative Networking

Verification in FVN
  NDLog Program Verification
  Verified Code Generation

Meta-Theoretic Model
  Compositional Routing Algebra

Conclusion

Open issues
Correctness-By-Construction via Metarouting

- Metarouting, algebraic framework for routing protocol
  - Models BGP systems (today’s de facto Internet routing) with convergence guarantee
Correct-by-construction via Metarouting, Cntd

- Our contribution: Formalize fragment of Metarouting theory in FVN using PVS
  - Heavy and interesting use of PVS theory interpretation: mapping and declaration
  - Extend PVS specification logic with metarouting theory
- Our goal: Free network operator from the tedious low-level and trivial theory consistency checking by leveraging PVS specification language and proof environment
Background on BGP system

- **Internet**, network of Autonomous Systems (AS) administrated by Internet Service Provider (ISP)
- *Routing Protocol* computes reachability information
- Internet routing is a combination of Internal Gateway Protocol (IGP) and External Gateway Protocol (EGP)
- **Border Gateway Protocol (BGP)**
  - de facto Internet routing
- BGP is policy based, ISP can influence route decision for economical or performance reasons
  - *Import policies* select routes to accept
  - *Export policies* decide routes to be advertised
- BGP is **NOT** ideal: no convergence guarantee
Metarouting
Algebraic framework for modeling BGP systems

- Abstract routing algebra, mathematical model: \( A = \langle \Sigma, \preceq, \mathcal{L}, \oplus, \mathcal{O}, \phi \rangle \)

  - sorts \( \Sigma \) (paths), \( \mathcal{L} \) (links)
  - opns \( \preceq: \Sigma \times \Sigma \rightarrow bool \) (preference relation)
    \( \oplus: \mathcal{L} \times \Sigma \rightarrow \Sigma \) (label application function)
  - \( \mathcal{O} \): subset of \( \mathcal{L} \) (origination set)
  - \( \phi: \Sigma \) (prohibited path)

- axioms
  - \( \forall \alpha \in \Sigma - \{ \phi \} \quad \alpha \preceq \phi \) (Maximality)
  - \( \forall l \in \mathcal{L} \quad l \oplus \phi = \phi \) (Absorption)
  - \( \forall l \in \mathcal{L} \forall \alpha \in \Sigma \quad \alpha \preceq l \oplus \alpha \) (Monotonicity)
  - \( \forall l \in \mathcal{L} \forall \alpha, \beta \in \Sigma \quad \alpha \preceq \beta \implies l \oplus \alpha \preceq l \oplus \beta \) (Isotonicity)

- Base algebras, building blocks for shortest path routing, etc
- Lexical product for route selection, composition operator

Isotonicity and Monotonicity: sufficient conditions for protocol convergence
Overview of PVS theories

Abstract Routing Algebra

A (Source)

PVS Mapping

Base Algebras

I_1 \ \cdots \ \I_n

Composition Operators

O_1 \ \cdots \ \O_m

- **A**: uninterpreted source theory `routeAlgebra`
- **I_i**: interpreted theory instantiated from **A**
- **O_i**: PVS theory taking routing algebra theories as parameters
Abstract Routing Algebra in PVS

routeAlgebra: THEORY
BEGIN
  sig: TYPE+
  label: TYPE+
END routeAlgebra
Abstract Routing Algebra in PVS

routeAlgebra: THEORY
BEGIN
  sig: TYPE+
  label: TYPE+
  injected: [label -> bool]
  org: TYPE = {l: label | injected(l)}
  prohibitPath: sig
  labelApply: [label, sig -> sig]
  prefRel: [sig, sig -> bool]
  eqRel(s1, s2: sig): bool = prefRel(s1, s2) \& prefRel(s2, s1)
  mono(l: label, s: sig): bool = prefRel(s, labelApply(l, s))
END
Abstract Routing Algebra in PVS

\textbf{routeAlgebra:} THEORY
BEGIN

\textbf{sig:} TYPE+
\textbf{label:} TYPE+
injected: [label \rightarrow bool]
\textbf{org:} TYPE = \{l: label \mid \text{injected}(l)\}
\textbf{prohibitPath:} sig
\textbf{labelApply:} [label, sig \rightarrow sig]
\textbf{prefRel:} [sig, sig \rightarrow bool]

eqRel(s_1, s_2: sig): bool = \text{prefRel}(s_1, s_2) \land \text{prefRel}(s_2, s_1)
mono(l: label, s: sig): bool = \text{prefRel}(s, \text{labelApply}(l, s))

\textbf{pref\_complete:} AXIOM
\forall (x, y: sig): \text{prefRel}(x, y) \lor \text{prefRel}(y, x)

\textbf{absorption:} AXIOM
\forall (l: label): \text{labelApply}(l, \text{prohibitPath}) = \text{prohibitPath}

\textbf{maximality:} AXIOM \forall (s: sig): \text{prefRel}(s, \text{prohibitPath})

\textbf{monotonicity:} AXIOM \forall (l: label, s: sig): \text{mono}(l, s)

\textbf{isotonicity:} AXIOM
\forall (s_1, s_2: sig)(l: label):
\begin{align*}
\text{prefRel}(s_1, s_2) & \Rightarrow \\
\text{prefRel}(\text{labelApply}(l, s_1), \text{labelApply}(l, s_2))
\end{align*}
END routeAlgebra
Motivation

Overview

Background on Declarative Networking

Verification in FVN

NDLog Program Verification

Verified Code Generation

Meta-Theoretic Model

Compositional Routing Algebra

Conclusion

Open issues
Base Algebra for Shortest Path Routing

PVS mapping: Abstract Algebra routeAlgebra → Base Algebra addA

- PVS mapping makes instantiations of uninterpreted types

  \[
  \begin{align*}
  \text{sig} & \leftarrow \text{upto}(m + 1) \\
  \text{label} & \leftarrow \text{upto}(n) \\
  \text{prohibitPath} & \leftarrow m + 1 \\
  \text{labelApply} & \leftarrow \text{APPLY} \\
  \text{prefRel} & \leftarrow \text{PREF}
  \end{align*}
  \]

- PVS mapping generates instances of routeAlgebra axioms as Type Correctness Conditions (TCCs)

  IMP_A_monotonicity_TCC1: OBLIGATION
  \[
  \text{FORALL } (l: \text{LABEL}, s: \text{SIG}): \text{mono}(l, s)
  \]
addA: THEORY
BEGIN
  n: posnat
  m: posnat
  redundant: posnat
N_M: AXIOM n < m
LABEL: TYPE = upto(n)
SIG: TYPE = upto(m + 1)
PREF(s₁, s₂: SIG): bool = (s₁ ≤ s₂)
APPLY(l: LABEL, s: SIG): SIG =
  IF (l + s < m + 1)
  THEN (l + s)
  ELSE (m + 1)
ENDIF
IMPORTING routeAlgebra
  {{sig := SIG, label := LABEL, prohibitPath := m + 1,
    labelApply(l: LABEL, s: SIG) := APPLY(l, s),
    prefRel(s₁, s₂: SIG) := (s₁ ≤ s₂)}}
END addA
Base Algebra for Provider-Customer, Peer-Peer Guideline

- For economical reasons, ISP reduces use of provider routes, and maximizes availability of customer routes

- $\Sigma(path)$: $C/R/P$ (customer/peer/provider path)

- $\mathcal{L}(link)$: $c/r/p$ (customer/peer/provider link)

- $\oplus$ (label application):

  \[
  \begin{array}{l|ccc}
    \oplus & C & R & P \\
    \hline
    c & C & C & C \\
    r & R & R & R \\
    p & P & P & P \\
  \end{array}
  \]

- $\preceq$ (preference relation): $C \preceq R$, $R \preceq P$, $C \preceq P$
For simplicity, rename labels and signatures:

\[ \begin{align*}
    c & \leftarrow 1, \\
    r & \leftarrow 2, \\
    p & \leftarrow 3 \quad \text{and} \quad C \leftarrow 1, \\
    R & \leftarrow 2, \\
    P & \leftarrow 3
\end{align*} \]

\[\text{lpA: THEORY}
\begin{align*}
\text{BEGIN} \\
\text{SIG: TYPE } &= \text{ upto(3)} \\
\text{LABEL: TYPE } &= \text{ upto(3)} \\
\text{IMPORTING routeAlgebra} \\
\{\{\text{sig } := \text{ SIG, label } := \text{ LABEL,} \\
\text{labelApply}(l: \text{ LABEL, s: SIG}) := l, \\
\text{prefRel}(s_1, \ s_2: \text{ SIG}) := (s_1 \leq s_2),\} \}
\text{END lpA}\]
Lexical Product $\otimes$ and Route Selection

- Lexicographic comparison models route selection
  - Most important attribute of each route is compared first, if no decision is reached, the next attribute is considered

- Lexical Product $A \otimes B$ built from existing algebras: $A, B$
  - Models a routing protocol with multiple attributes
  - More important attributes are handled by $A$, and the less important by $B$
Lexical Product $A \otimes B$ in PVS

PVS declaration and mapping ensures resulting algebra $A \otimes B$ is a valid routing algebra, i.e. $\otimes$ is closed under abstract routing algebra.

```
lexProduct[A: THEORY routeAlgebra, B: THEORY routeAlgebra]: THEORY
BEGIN

SIG: TYPE = [A.sig, B.sig]

LABEL: TYPE = [A.label, B.label]

APPLY(l: LABEL, s: SIG): SIG =
  (A.labelApply(l'1, s'1), B.labelApply(l'2, s'2))

PREF(s1, s2: SIG): bool =
  A.prefRel(s1'1, s2'1) \lor
  (A.eqRel(s1'1, s2'1) \land B.prefRel(s1'2, s2'2))

IMPORTING routeAlgebra
  {{sig := SIG, label := LABEL,
    labelApply(l: LABEL, s: SIG) := APPLY(l, s),
    prefRel(s1, s2: SIG) := PREF(s1, s2)}}

END lexProduct
```
A Concrete BGP system

- Route paths are measured in terms of customer-provider relationship and distance cost
  - Customer-Provider Peer-Peer guideline must be enforced
  - Once customer-provider policy is satisfied, ISP wants least-cost (shortest) paths
- Decompose this BGP system into two sub-components
  - Sub-component A for customer-provider guideline
  - Sub-component B for shortest-path
  - Check the sub-component A first, and only use B to break tie
- $\text{BGPsystem: THEORY}=\text{lexProduct}[A_2, B_2]$
Concrete BGP system in PVS
Sub-components Instantiated from Base Algebras

AlgebraInstance: THEORY
BEGIN

IMPORTING addA\{n := 16, m := 16\}

IMPORTING lpA\{c := 3\}

A_2: THEORY = 
routeAlgebra
\{\{\text{sig} = \text{lpA}.SIG, \text{label} = \text{lpA}.LABEL, 
\text{labelApply}(l: \text{lpA}.LABEL, s: \text{lpA}.SIG) = l + s, \text{prohibitPath} = 4, 
\text{prefRel}(s_1, s_2: \text{int}) = (s_1 \leq s_2)\}\}

B_2: THEORY = 
routeAlgebra
\{\{\text{sig} = \text{addA}.SIG, \text{label} = \text{addA}.LABEL, 
\text{labelApply}(l: \text{addA}.LABEL, s: \text{addA}.SIG) = \text{mod}(l + s, 16), 
\text{prohibitPath} = 17, 
\text{prefRel}(s_1, s_2: \text{addA}.SIG) = (s_1 \leq s_2)\}\}

END AlgebraInstance
Motivation

Overview
  Background on Declarative Networking

Verification in FVN
  NDLog Program Verification
  Verified Code Generation

Meta-Theoretic Model
  Compositional Routing Algebra

Conclusion

Open issues
Design: correctness-by-construction via meta-model
Conclusion, Recap

- **Design**: correctness-by-construction via meta-model
- **Specification**: two way property preserving translation
  - Formal system specification generated from NDlog program *(arc 4)*
  - Executable Declarative network synthesized from verified logical specification *(arc 3)*
Conclusion, Recap

- **Design**: correctness-by-construction via meta-model
- **Specification**: two way property preserving translation
  - Formal system specification generated from NDlog program *(arc 4)*
  - Executable Declarative network synthesized from verified logical specification *(arc 3)*
- **Verification**: proving network invariants of system specifications by interacting with theorem prover *(arc 5)*
Conclusion, Recap

- **Design**: correctness-by-construction via meta-model
- **Specification**: two way property preserving translation
  - Formal system specification generated from NDlog program (arc 4)
  - Executable Declarative network synthesized from verified logical specification (arc 3)
- **Verification**: proving network invariants of system specifications by interacting with theorem prover (arc 5)
- **Implementation**: distributed query processing (arc 7)
Motivation

Overview
  Background on Declarative Networking

Verification in FVN
  NDLog Program Verification
  Verified Code Generation

Meta-Theoretic Model
  Compositional Routing Algebra

Conclusion

Open issues
Open Issues
Destructive Updates, Linear Logic, Model Checking

- Non-monotonic updates in Networking and security
  - Link-failure, base tuples removed from routing table
  - Incremental maintenance of derived tuples
- Classical/Intuitive logic is monotonic
  finner control with linear logic?
  - Logical premises consumed (tuples removed) after use in proof/derivation
  - Linear logic as the semantic foundation for Declarative Networking with soft-state?
Open Issues, Cntd
Exploring Various Models, Verification Techniques

- Network Models and Implementation
  - Relaxed Algebraic models for wider range of protocols
    - Metarouting algebra are monotonic
    - Non-monotonic attributes provide useful semantic in real networking, e.g. MED.
  - Alternative component-based models: Click, Xorp

- Combining Verification Techniques
  - Theorem proving, declarative networking specific decision procedures/proof strategies/tactics
  - Model checking declarative networking, transitions updating routing tables are specified in linear logic
  - Automatically model checking finite small networking case, and using theorem proving to scale
Thank You!

http://netdb.cis.upenn.edu/fvn/