Automated Verification of Safety Properties of Declarative Networking Programs

Chen Chen¹, Lay Kuan Loh², Limin Jia³, Wenchao Zhou³, Boon Thau Loo¹
¹University of Pennsylvania, ²Carnegie Mellon University, ³Georgetown University

INTRODUCTION

Motivation
- Networks are complex systems that unfortunately are ridden with errors
- Such errors can lead to disruption of services with grave consequences

Our Solution
- Network Verification
  - Encode the network protocol in Network Datalog
  - Recursive query language over network states
  - Each entry in the derivation pool maps to a distinct tuple in the NDLog program

Experimental
- We tested our tool on four network applications: ethernet source learning, load balancer, firewall, and address resolution protocol. Each case study ran to completion within 1 second.

Formal specification of networks in a declarative language

- Specify networks in a declarative language
- Specify property in first-order logic
- Automated verification of safety properties

Concrete faulty execution traces when proof fails

DGraph

- Rule Head → Body, Body₀, …, Bodyₙ, Constraint
@: Location specifier

Dependency Graph

Vertices
- Tuple nodes
- Rule nodes

Edges
- [Rule node → Head tuple node]
- [Body tuple node → Rule node]

Example dependency graph for Twohops

```
Twohops
R1 onehop(@z,x,c2) :- link(@z,x,c2)
R2 twohops(@x,y,c) :- onehop(@z,x,c2), onehop(@z,x,c2), c=c1+c2
```

GenDPool

- Derivation pool construction
- Each entry in the derivation pool maps to a distinct tuple in the NDLog program
- Consists of list of possible derivations and their corresponding constraints

Specify safety property in first-order logic

Safety property: Something bad never happens

Restricted property format

```
∀ x₁,p₁(x₁) \& \& ∀ x₂,p₂(x₂) \& \& … \& \& ∀ xₙ,pₙ(xₙ) \& \& c₁(x₁, …, xₙ) \& \& \& y₁, q₁(y₁) \& \& y₂, q₂(y₂) \& \& … \& \& yₘ, qₘ(yₘ) \& \& c₂(x₁, …, xₙ, y₁, …, yₘ)
```

Example invalid safety property for Twohops

If the cost of traversing a onehop tuple is greater than zero, then there exists a link tuple, such that the cost of traversing that link tuple is less than zero

```
∀ x₁,x₂,x₃, onehop(x₁,x₂,x₃) \& \& (c₁(x₁, x₂, x₃) \& \& y₁, y₂, y₃) \& \& (y₃ < 0)
```

METHODOLOGY OUTLINE

Formal specification of networks in a declarative language

- DGraph
- Dependancy Graph
- GenDPool

Property Verification

- Proved correctness
- Valid
- Invalid

Produce concrete counterexample

```
∀ x₁,x₂,x₃, onehop(x₁,x₂,x₃) \& \& (c₁(x₁, x₂, x₃) \& \& y₁, y₂, y₃) \& \& (y₃ < 0)
```

Example counterexample construction for Twohops

```
l1(x₁,x₂,x₃) \& \& (y₁ = 1, y₂ = y₃) \& \& (y₃ < 0)
```

Verify the property holds for all possible derivations of tuples in the antecedent of the property

```
∀ x₁,x₂,x₃, onehop(x₁,x₂,x₃) \& \& (c₁(x₁, x₂, x₃) \& \& y₁, y₂, y₃) \& \& (y₃ < 0)
```

```
l1(x₁,x₂,x₃) \& \& (y₁ = 1, y₂ = y₃) \& \& (y₃ < 0)
```

Verify validity of constraints by using an SMT solver

```
l1(x₁,x₂,x₃) \& \& (y₁ = 1, y₂ = y₃) \& \& (y₃ < 0)
```

```
l1(x₁,x₂,x₃) \& \& (y₁ = 1, y₂ = y₃) \& \& (y₃ < 0)
```

```
l1(x₁,x₂,x₃) \& \& (y₁ = 1, y₂ = y₃) \& \& (y₃ < 0)
```

```
l1(x₁,x₂,x₃) \& \& (y₁ = 1, y₂ = y₃) \& \& (y₃ < 0)
```

```
l1(x₁,x₂,x₃) \& \& (y₁ = 1, y₂ = y₃) \& \& (y₃ < 0)
```

COMPLEXITY

Theoretical
- Given an NDLog program with R rules where each rule has at most W body tuples, and a property where n \& predicates in the antecedent, m \& predicates in the consequent, then the time complexity is O(R(W^n+m)^Wn).

Experimental
- We tested our tool on four network applications: ethernet source learning, load balancer, firewall, and address resolution protocol. Each case study ran to completion within 1 second.