Automated Verification of Safety Properties of Declarative Networking Programs

Chen Chen[1], Lay Kuan Loh[2], Limin Jia[2], Wenchao Zhou[3], Boon Thau Loo[1]

[1]University of Pennsylvania, [2]Carnegie Mellon University, [3]Georgetown University

Penn | Carnegie Mellon University | GEORGETOWN 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

Specify networks in a declarative language → Automated verification of safety properties

Specify property in first-order logic → Automated verification of safety properties

Automated verification of safety properties → Valid / Invalid

Invalid → Concrete faulty execution traces when proof fails

METHODOLOGY OUTLINE

Formal specification of networks in a declarative language

↓

DGraph

↓

Dependency Graph

↓

GenDPool

↓

Derivation pool construction (Proved correctness)

Specify safety property in first-order logic → Property Verification

Derivation pool construction → Property Verification (Proved correctness)

Property Verification → Valid / Invalid

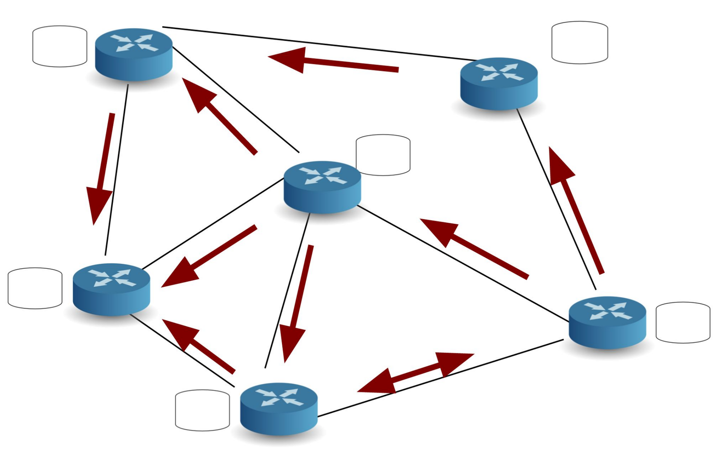
Invalid → Produce concrete counterexample

Formal specification of networks in a declarative language

- Encode the network protocol in Network Datalog (NDLog), a distributed variant of Datalog
- Recursive query language over network states

Rule Head :- $Body_1$, $Body_2$, …, $Body_n$, Constraint

@: Location specifier



DGraph

Dependency Graph

NDLog Program ⇒ **Vertices**: Tuple nodes, Rule nodes; **Edges**: [Rule node → Head tuple node], [Body tuple node → Rule node]

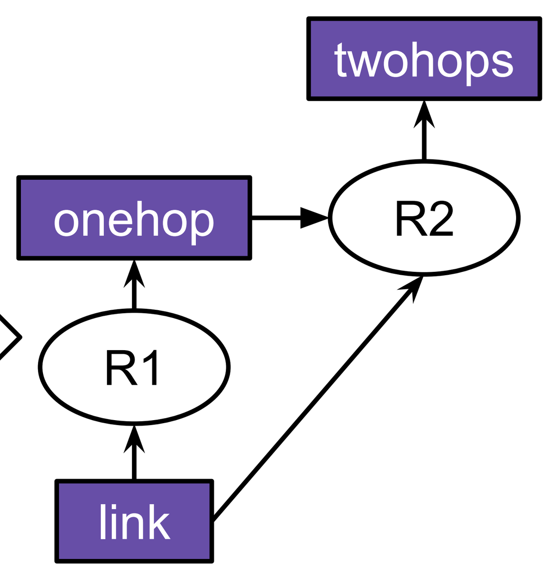
Example dependency graph for Twohops

Twohops

R1 onehop(@z,x,c2) :- link(@z,x,c2)

R2 twohops(@x,y,c) :-

link(@z,y,c1), 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 "◆" Indicates the temporal operation "past"

$$\forall x_1.p_1(x_1) \wedge \forall x_2.p_2(x_2) \wedge \ldots \wedge \forall x_n.p_n(x_n) \wedge c_q(x_1,\ldots,x_n) \supset \exists y_1.\blacklozenge q_1(y_1) \wedge \exists y_2.\blacklozenge q_2(y_2) \wedge \ldots \wedge \exists y_m.\blacklozenge q_m(y_m) \wedge c_q(x_1,\ldots,x_n,y_1,\ldots,y_m)$$

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

$$\forall x_1,x_2,x_3.onehop(x_1,x_2,x_3) \wedge (x_3>0) \supset \exists y_1,y_2,y_3.\blacklozenge link(y_1,y_2,y_3) \wedge (y_3<0)$$

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 consequent, then the time complexity is $\mathbf{O((R^{nW^{\wedge}R})n^mW^{Rn})}$.

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.

Property Verification

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

↓

For each combination of derivations of tuples in the antecedent of the property

→ Verify existence of tuples in the consequent by ***looking for instances of tuples in the given derivation*** → ✔ Valid / ✘ Invalid

→ Verify validity of constraints by ***using an SMT solver*** → ✔ Valid / ✘ Invalid

Produce concrete counterexample

Find a satisfying substitution for the negation of the constraints to generate a concrete counterexample

Example counterexample construction for Twohops

$$\forall x_1,x_2,x_3.onehop(x_1,x_2,x_3) \wedge (x_3>0) \supset \exists y_1,y_2,y_3.\blacklozenge link(y_1,y_2,y_3) \wedge (y_3<0)$$

Verify the property holds for all possible derivations of tuples in the antecedent
Tuple **onehop** in the antecedent has only one possible derivation

$link(z_1,z_2,z_3)$ → $onehop(x_1,x_2,x_3)$

Variable unification: $x_3=z_3 \wedge y_3=z_3$

Verify validity of constraints
$CheckSAT((x_3=z_3 \wedge y_3=z_3) \wedge (x_3>0) \wedge \neg(y_3<0))$

→ UNSAT / SAT

SAT → *Construct counterexample*
$x_3, z_3 = 0, y_3 = 1$ satisfies
$\neg((x_3=z_3 \wedge y_3=z_3) \wedge (x_3>0) \wedge (y_3<0))$

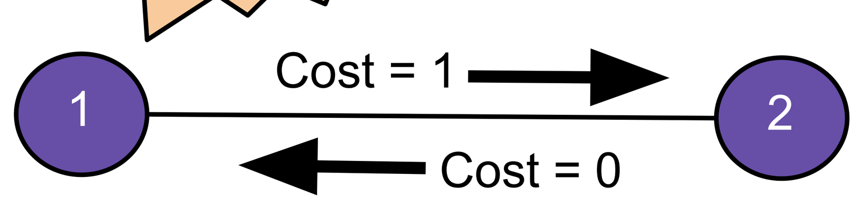
Verify existence of tuples in the consequent
Tuple **link** appears in the derivation of **onehop**

link(@1,2,1) → onehop(@2,1,0)

Link symmetry violated!

link

@Src	Dst	Cost
@1	2	1



onehop

@Src	Dst	Cost
@2	1	0