Program Analysis for System Security and Reliability

Petar Tsankov
Spring 2018

http://www.srl.inf.ethz.ch
http://ice.ethz.ch

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich
Last lecture

Verifying existing network configurations

- **Basic concepts**
  - Partially-ordered sets, complete lattices
  - Monotone functions, fixed points

- **Datalog**
  - Consequence operator, fixed-point semantics
  - Practical framework for security analysis of networks, smart contracts

- **Network verification**
  - Network configurations ➔ Datalog inputs
  - Routing protocols ➔ Datalog program
  - Forwarding plane ➔ Datalog fixed-point
Today

How do we come up with a correct configuration?

Synthesizing correct network configurations

Network-wide Configuration Synthesis, CAV 2017
Ahmed El-Hassany, Petar Tsankov, Laurent Vanbever, Martin Vechev

https://synet.ethz.ch

Network-wide Configuration Synthesis, NSDI 2018
Ahmed El-Hassany, Petar Tsankov, Laurent Vanbever, Martin Vechev

https://netcomplete.ethz.ch
Software synthesis @ SRL

Develop new synthesis techniques to solve practical system challenges

Intent  Desired properties

Constraint-based
CEGIS
Probabilistic

Big Code
Oracle-guided

Software Synthesis

Computer networks  Security and privacy  Modern architectures  Datacenters  Data science  End-user programming
A note on program synthesis

Program Synthesis is one of the most exciting recent developments in artificial intelligence. It focuses on one of computer science’s most exciting visions: learning a function from specifications (e.g., examples).

The main focus of today’s program synthesis vs. that of decades ago is the incorporation of powerful SAT/SMT solvers, multiple ways to convey intent (e.g., labeled data).

How does program synthesis fit in the domain of networks?
Current practice

Initially not configured

Network topology

Routing requirements

Reachability, isolation, ...

Operators manually configure each router

Network with configured routers
Wanted: Programmable networks with synthesis

Network topology

Routing requirements

Operators manually configure each router

How to find a configuration that conforms to the requirements?

Network with configured routers
Programmable networks with synthesis: Dimensions

- Deployment scenarios
  - Datacenter
  - ISP
  - Incremental
  - Enterprise

- Routing protocols
  - OSPF
  - BGP
  - ECMP
  - Static routes
  - MPLS
  - Gossip

- Requirements
  - Paths
  - Reachability
  - Congestion
  - Isolation
  - Failures
  - Waypointing

- Synthesis Techniques
  - Enumerative learning
  - Probabilistic
  - Symbolic execution
  - CEGIS
  - Constraint solving

SyNET
# Programmable Networks with Synthesis: Dimensions

## Deployment scenarios
- Datacenter
- ISP
- Incremental
- Enterprise

## Routing protocols
- OSPF
- BGP
- ECMP
- MPLS
- Static routes
- Gossip

## Requirements
- Paths
- Reachability
- Isolation
- Waypointing
- Congestion
- Failures

## Synthesis Techniques
- Enumerative learning
- Probabilistic
- Symbolic execution
- Constraint solving
- CEGIS

**SyNET**

**NetComplete**
Batfish: Analysis of Network Configurations

Network configuration

Distributed protocols (BGP, OSPF, Static routes)

Requirements (isolation, reachability)

Does the configured network satisfy the requirements?

Datalog input I

Datalog program P

Query q

$q \in \text{lfp}_{T_{PUI}}$
Plan for Today

I. Background
- SMT / Symbolic execution / Input synthesis

II. SyNET
- Input synthesis for Datalog
- Scalability of SyNET

III. Efficient OSPF synthesis
- Counter-Example Guided Inductive Synthesis (CEGIS)
- CEGIS vs direct synthesis for OSPF
Symbolic execution

In classic symbolic execution, we associate with each variable a symbolic value instead of a concrete value. We then run the program with the symbolic values obtaining a big constraint formula as we run the program. Hence, the name symbolic execution.

At any program point we can invoke a constraint (SMT) solver to find satisfying assignments to the formula. These satisfying assignments can be used to indicate concrete inputs for which the program reaches a program point or to steer the analysis to another part of the program.
Constraint examples

Linear constraint: \(5 \times x + 6 < 100\)

Non-linear constraint: \(x \times y + 12 < 29\)

A constraint solver, typically an SMT solver, finds satisfying assignments to constraints. An example of SMT solvers are Z3 and Yices.

Demo: [https://rise4fun.com/Z3](https://rise4fun.com/Z3)
Symbolic execution overview

At any point during program execution, symbolic execution keeps two formulas:

symbolic store and path constraint

Tracks possible values of variables

Tracks history of all branches taken so far

Therefore, at any point in time the symbolic state is described as the conjunction of these two formulas.
Symbolic execution example

```c
int twice(int v) {
    return 2 * v;
}

void test(int x, int y) {
    z = twice(y);
    if (x == z) {
        if (x > y + 10)
            ERROR;
    }
}

int main() {
    x = read();
    y = read();
    test(x, y);
}
```

Find an input that causes the program to reach the ERROR?

Symbolic store:
- $x \mapsto x_0$
- $y \mapsto y_0$

Path constraint:
- true
Symbolic execution example

```c
int twice(int v) {
    return 2 * v;
}

void test(int x, int y) {
    z = twice(y);
    if (x == z) {
        if (x > y + 10)
            ERROR;
    }
}

int main() {
    x = read();
    y = read();
    test(x,y);
}
```

Find an input that causes the program to reach the **ERROR**?

- **Symbolic store**
  - $x \mapsto x_0$
  - $y \mapsto y_0$
  - $z \mapsto 2 \times y_0$

- **Path constraint**
  - **true**

We encounter a branch: we fork the execution along both branches (next we proceed with the true branch).
Symbolic execution example

```c
int twice(int v) {
    return 2 * v;
}

void test(int x, int y) {
    z = twice(y);
    if (x == z) {
        if (x > y + 10)
            ERROR;
    }
}

int main() {
    x = read();
    y = read();
    test(x, y);
}
```

Find an input that causes the program to reach the ERROR?

Symbolic store:
- $x \mapsto x_0$
- $y \mapsto y_0$
- $z \mapsto 2 \times y_0$

Path constraint:
- $x_0 = 2 \times y_0$

We update the path constraint to capture that the $(x == z)$ evaluates to true.
Symbolic execution example

```c
int twice(int v) {
    return 2 * v;
}

void test(int x, int y) {
    z = twice(y);
    if (x == z) {
        if (x > y + 10)
            ERROR;
    }
}

int main() {
    x = read();
    y = read();
    test(x, y);
}
```

Find an input that causes the program to reach the **ERROR**?

Symbolic store

- $x \mapsto x_0$
- $y \mapsto y_0$
- $z \mapsto 2 \times y_0$

Path constraint

- $x_0 = 2 \times y_0$
- $x_0 > y_0 + 10$

We can now ask the SMT solver for a satisfying assignment to the path constraint.

For instance, $x_0 = 40, y_0 = 20$ is a satisfying assignment. That is, running the program with these inputs triggers the error.
Symbolic execution in practice

- Many challenges (not covered in this lecture):
  - Loops
  - Non-linear constraints
  - Hard-to-solve constraints ($x = \text{hash}(y)$)

- Heavily used in practice to find security bugs (overflows, memory errors, etc.) in real software
  - Stanford’s KLEE: [http://klee.doc.ic.ac.uk/](http://klee.doc.ic.ac.uk/)
  - Microsoft Research’s SAGE
  - UC Berkeley’s CUTE
  - EPFL’s S2E: [http://dslab.epfl.ch/proj/s2e](http://dslab.epfl.ch/proj/s2e)
Plan for Today

I. Background
- SMT / Symbolic execution / Input synthesis

II. SyNET
- Input synthesis for Datalog
- Scalability of SyNET

III. Efficient OSPF synthesis
- Counter-Example Guided Inductive Synthesis (CEGIS)
- CEGIS vs direct synthesis for OSPF
Recall: Synthesis of network configurations

Distributed protocols (BGP, OSPF, Static routes)

Requirements (isolation, reachability)

Find a configuration that satisfies the requirements

Network configuration

Datalog program \( P \)

Query \( q \)

Find and an input \( I \) such that:

\[ q \in \text{lfp}_{TPUI} \]

Datalog input \( I \)
Programs vs Datalog

Symbolic execution can be used to synthesize an input that steers the execution along a specific path in a program.

Can we synthesize an input that steers a Datalog program to derive a specific atom?

We need to solve the following problem:

Given a Datalog program $P$ and a query $q$, find an input $I$ such that: $q \in \text{lfp}_{T_{PUI}}$
Basic example from last lecture

Input
link(c2, n1) <-
link(n1, n2) <-
link(n2, n10) <-
...

Program
path(X, Y) <- link(X,Y)
path(X, Y) <- link(X, Z), path(Z, Y)

Query
path(c2, n10)
!link(c2, n10)
Input synthesis for Datalog

Input
link(?, ?) <-
link(?, ?) <-
link(?, ?) <-
...

Program
path(X, Y) <- link(X,Y)
path(X, Y) <- link(X, Z), path(Z, Y)

Query
path(c2, n10)
!link(c2, n10)
Datalog rules as SMT constraints

Datalog rules are logical constraints that can be fed to an SMT solver.

By Datalog semantics, variables in the head of a rule are quantified universally and those in the body are quantified existentially. For example,

\[
\text{path}(X,Y) \leftarrow \text{link}(X,Z), \text{path}(Z,Y)
\]

becomes:

\[
\forall X,Y. \text{path}(X,Y) \leftarrow (\exists Z. \text{link}(X,Z) \land \text{path}(Z,Y))
\]

where the variables \(X, Y, Z\) range over a fixed domain \(\{n1, n2, \ldots\}\).

Can we directly use an SMT solver to generate an input?
Input synthesis for Datalog via SMT

Key steps:

1. **Encode** $P$ into SMT constraints, which capture the fixed-point computed by $P$ for a given input $I$
2. **Encode** $q$ as assertions that must hold on the fixed-point
3. Get a model $M$ that satisfies the conjunction of the above constraints
4. **Derive input** $I$ from $M$ by checking which atoms are true in $M$
Input synthesis for positive Datalog (first attempt)

Program
path(X, Y) <- link(X,Y)
path(X, Y) <- link(X, Z), path(Z, Y)

Query
path(c2, n10)
!link(c2, n10)

Generate SMT constraints

SMT Constraints $\psi$

∀X,Y. path(X, Y) ⇐ link(X, Z)
∀X,Y. path(X, Y) ⇐ (∃Z. link(X, Z) ∧ path(Z, Y))

A first attempt in encoding the Datalog rules into SMT constraints
Input synthesis for positive Datalog (first attempt)

Program
\[
\begin{align*}
\text{path}(X, Y) & \leftarrow \text{link}(X, Y) \\
\text{path}(X, Y) & \leftarrow \text{link}(X, Z), \text{path}(Z, Y)
\end{align*}
\]

Query
\[
\begin{align*}
\text{path}(1, 2) \\
!\text{link}(1, 2)
\end{align*}
\]

Generate SMT constraints

SMT Constraints \( \psi \)
\[
\begin{align*}
\forall X, Y. \text{path}(X, Y) & \leftarrow \text{link}(X, Z) \\
\forall X, Y. \text{path}(X, Y) & \leftarrow (\exists Z. \text{link}(X, Z) \land \text{path}(Z, Y)) \\
\text{path}(1, 2) \land \neg \text{link}(1, 2)
\end{align*}
\]

Constraint that captures the query

Demo: [https://rise4fun.com/Z3](https://rise4fun.com/Z3)
Input synthesis for positive Datalog (first attempt)

Program
path(X, Y) ← link(X,Y)
path(X, Y) ← link(X, Z), path(Z, Y)

Query
path(1, 2)
!link(1, 2)

Generate SMT constraints

SMT Constraints $\psi$

$\forall X, Y. \text{path}(X, Y) \leftarrow \text{link}(X, Z)$
$\forall X, Y. \text{path}(X, Y) \leftarrow (\exists Z. \text{link}(X, Z) \land \text{path}(Z, Y))$
path(1,2) $\land \neg \text{link}(1, 2)$

Find a model

Model $M = \{\text{path}(1,2), \text{path}(1,3)\}$

Derive input

Input $I = \{\}$

Unfortunately, path(1,2) $\not\in \text{lfp}_{PUI}$
What went wrong?

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>Q</td>
<td>P ⇒ Q</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

- In a Datalog, given a rule $p ← q$, $p$ is derived iff $q$ is true
- In logic, the constraint $p \iff q$ is satisfied if $p$ is true and $q$ is false

This means that not every model of the Datalog rules uniquely captures the fixed-point of the Datalog program.
Bounded unrolling of Datalog

Program
\[ \text{path}(X, Y) \leftarrow \text{link}(X, Y) \]
\[ \text{path}(X, Y) \leftarrow \text{link}(X, Z), \text{path}(Z, Y) \]

Query
\[ \text{path}(1, 2) \]
\[ !\text{link}(1, 2) \]

Generate SMT constraints

SMT Constraints $\psi$
\[ \forall X, Y. \text{path}_1(X, Y) \iff \text{link}(X, Z) \]

$\text{path}_1(X, Y)$ captures all $\text{path}(X, Y)$ atoms derived after 1 step of the consequence operator
Bounded unrolling of Datalog

Program
path(X, Y) ← link(X,Y)
path(X, Y) ← link(X, Z), path(Z, Y)

Query
path(1, 2)
!link(1, 2)

Generate SMT constraints

SMT Constraints ψ
∀X, Y. path₁(X, Y) ⇔ link(X, Z)
∀X, Y. path₂(X, Y) ⇔ (link(X, Y) ∨ (∃Z. link(X, Z) ∧ path₁(Z, Y)))

path₂(X, Y) captures all path(X, Y) atoms derived after 2 steps of the consequence operator
Bounded unrolling of Datalog

Program
path(X, Y) ← link(X,Y)
path(X, Y) ← link(X, Z), path(Z, Y)

Query
path(1, 2)
!link(1, 2)

Generate SMT constraints

SMT Constraints $\psi$

$\forall X, Y. path_1(X,Y) \iff link(X,Z)$
$\forall X, Y. path_2(X,Y) \iff (link(X,Y) \lor (\exists Z. link(X,Z) \land path_1(Z,Y)))$
path_2(1,2) \land \neg link(1,2)

Demo: [https://rise4fun.com/Z3](https://rise4fun.com/Z3)
Bounded unrolling of Datalog

Program
path(X, Y) <- link(X,Y)
path(X, Y) <- link(X, Z), path(Z, Y)

Query
path(1, 2)
!link(1, 2)

Generate SMT constraints

SMT Constraints $\psi$

$\forall X, Y. \text{path}_1(X, Y) \Leftrightarrow \text{link}(X, Z)$
$\forall X, Y. \text{path}_2(X, Y) \Leftrightarrow (\text{link}(X, Y) \lor (\exists Z. \text{link}(X, Z) \land \text{path}_1(Z, Y)))$
$\text{path}_2(1,2) \land \neg \text{link}(1,2)$

Find a model

Model $M = \{\text{path}(1,2), \text{path}(1,3), \text{path}(3,2), \text{link}(1,3), \text{link}(3,2)\}$

Derive input

Input $I = \{\text{link}(1,3), \text{link}(3,2)\}$
What about negative queries?

Unrolling the Datalog rules works for positive queries because, by monotonicity of the consequence operator, any atom derived in a given step of the consequence operator is guaranteed to remain in the fixed point.

What if the query requires that certain atoms are not contained in the fixed point?
Handling negative queries

Program
path(X, Y) <- link(X, Y)
path(X, Y) <- link(X, Z), path(Z, Y)

Query
!path(1, 3)

Generate SMT constraints

SMT Constraints ψ
∀X, Y. path(X, Y) ⇐ link(X, Z)
∀X, Y. path(X, Y) ⇐ (∃Z. link(X, Z) ∧ path(Z, Y))
¬path(1, 3)

Find a model
Model M = {}

Derive input
Input I = {}
**Input synthesis for Datalog: Final solution**

<table>
<thead>
<tr>
<th>path(X, Y) &lt;- link(X, Y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>path(X, Y) &lt;- link(X, Z), path(Z, Y)</td>
</tr>
</tbody>
</table>

| path(1, 2), !link(1, 2), !path(1, 3) |

∀X, Y. \( path_1(X, Y) \Leftrightarrow \text{link}(X, Z) \)
∀X, Y. \( path_2(X, Y) \Leftrightarrow (\text{link}(X, Y) \lor (\exists Z. \text{link}(X, Z) \land path_1(Z, Y)) \)
...
∀X, Y. \( path_n(X, Y) \Leftrightarrow (\text{link}(X, Y) \lor (\exists Z. \text{link}(X, Z) \land path_{n-1}(Z, Y)) \)

∀X, Y. \( path(X, Y) \Leftarrow \text{link}(X, Z) \)
∀X, Y. \( path(X, Y) \Leftarrow (\exists Z. \text{link}(X, Z) \land path(Z, Y)) \)

\( \neg path_n(1,2) \land \neg \text{link}(1, 2) \land \neg path(1,3) \)

**Bounded unrolling for positive queries**

**No unrolling for negative queries**
Input synthesis for stratified Datalog

Suppose we have a program $P$ with strata $P_1, \ldots, P_n$, and a query $q$.

High-level idea:
**Back step:** Backtrack to step Synth $P_i$ if the step Synth $P_{i-1}$ returns unsat.

**Synth $P_n$:** Compute input $I_n$ for stratum $P_n$ such that $[P_n]_{I_n}$ satisfies $q$.

**Synth $P_{n-1} \ldots P_1$:** Compute input $I_i$ for stratum $P_i$ such that $[P_i]_{I_i}$ produces the input $I_{i+1}$ synthesized by the previous step.
Recap: Synthesis of network configurations

Reduction to input synthesis for Datalog

Datalog input identifies correct configurations

Automatically configure routers with synthesis

Network with configured routers

Network topology

Routing requirements

How well does this work in practice?
Experiment

US-based network connecting major universities and research institutes

<table>
<thead>
<tr>
<th>Protocols / # Traffic classes</th>
<th>1 class</th>
<th>5 classes</th>
<th>10 classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>1.3s</td>
<td>2.0s</td>
<td>4.0s</td>
</tr>
<tr>
<td>Static + OSPF</td>
<td>9.0s</td>
<td>21.3s</td>
<td>49.3s</td>
</tr>
<tr>
<td>Static + OSPF + BGP</td>
<td>13.3s</td>
<td>22.7s</td>
<td>1m19.7s</td>
</tr>
</tbody>
</table>
Larger networks

Grid topologies with up to 64 routers
Requirements for 10 traffic classes

<table>
<thead>
<tr>
<th># of routers</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synthesis Time [s]</td>
<td>10^9</td>
<td>10^10</td>
<td>10^2</td>
<td>10^3</td>
<td>10^4</td>
<td>10^5</td>
</tr>
</tbody>
</table>

- Static
- Static + OSPF
- Static + OSPF + BGP
Larger networks

Grid topologies with up to 64 routers

Requirements for 10 traffic classes

Key challenge: Scaling synthesis for OSPF
Plan for Today

I. Background
   ▪ SMT / Symbolic execution / Input synthesis

II. SyNET
   ▪ Input synthesis for Datalog
   ▪ Scalability of SyNET

III. Efficient OSPF synthesis
   ▪ Counter-Example Guided Inductive Synthesis (CEGIS)
   ▪ CEGIS vs direct synthesis for OSPF
Requirement
For paths from src to dst:
- src → a → b → c → dst has the least cost
- src → a → b → d → c → dst is second least-cost

Define a constraint that captures the requirement
OSPF requirement

\[ P_1 = \text{src} \rightarrow a \rightarrow b \rightarrow c \rightarrow d \text{st} \]
\[ P_2 = \text{src} \rightarrow a \rightarrow b \rightarrow d \rightarrow c \rightarrow d \text{st} \]

\[ \text{Cost}(P_1) = c_{a,b} + c_{b,c} \]
\[ \text{Cost}(P_2) = c_{a,b} + c_{b,d} + c_{d,c} \]

Let \( \text{Paths} \) be the set of all simple paths from \( \text{src} \) to \( d \text{st} \) and \( C \) the set of all cost variables

Encoding the requirement:

\[ \psi(\text{Paths}, C) = \begin{cases} \text{Cost}(P_1) < \text{Cost}(P_2) \\ \land \forall X \in \text{Paths} \setminus \{P_1, P_2\}. \text{Cost}(P_2) < \text{Cost}(X) \end{cases} \]
Direct OSPF synthesis

Find cost assignment $A$ such that $\phi(\text{Paths}, A)$ holds

Formula given to the solver: $\exists C. \phi(\text{Paths}, C)$
Challenges with direct OSPF synthesis

The formula $\exists C. \phi(P, C)$ is hard to solve
- The constraint $\phi(P, C)$ quantifies over all simple paths in $P$
- There are exponentially many simple paths

Insight:
- A small set $E$ of paths can be sufficient to constraint the solution
  $\exists C. \phi(E, C)$, where $E = \{P_1, P_2, ..., P_k\} \subseteq P$
- How do we find the set $E$?
CEGIS = **Counter-Example Guided Inductive Synthesis**

**Inductive Synthesizer**

Derive candidate link costs from a subset $E$ of the paths

$\exists C. \phi(E, C)$

**Oracle (verifier)**

Check whether $\phi(P, C)$ holds over all paths in $P$

- **succeed**
- **fail**

**Candidate assignment $C$**

- **succeed**
- **fail**

**Observation set $E$**

- **ok**
- **unsat**

Add counterexample path to $E$
## Direct vs naïve OSPF synthesis

<table>
<thead>
<tr>
<th>Network size</th>
<th>Req. type</th>
<th>16 requirements</th>
<th>50% symbolic</th>
<th>100% symbolic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>CEGIS</td>
<td>Naive</td>
<td>CEGIS</td>
</tr>
<tr>
<td>Small</td>
<td>Simple</td>
<td>3.33</td>
<td>6.95</td>
<td>3.39</td>
</tr>
<tr>
<td></td>
<td>Any-path</td>
<td>4.63</td>
<td>7.22</td>
<td>4.76</td>
</tr>
<tr>
<td></td>
<td>ECMP</td>
<td>3.44</td>
<td>3.16</td>
<td>3.48</td>
</tr>
<tr>
<td></td>
<td>Ordered</td>
<td>4.76</td>
<td>5.07</td>
<td>7.93</td>
</tr>
<tr>
<td>Medium</td>
<td>Simple</td>
<td>6.17</td>
<td>3238.46</td>
<td>6.18</td>
</tr>
<tr>
<td></td>
<td>Any-path</td>
<td>9.70</td>
<td>107.13</td>
<td>9.83</td>
</tr>
<tr>
<td></td>
<td>ECMP</td>
<td>6.34</td>
<td>45.32</td>
<td>6.39</td>
</tr>
<tr>
<td></td>
<td>Ordered</td>
<td>31.08</td>
<td>49.43</td>
<td>43.63</td>
</tr>
<tr>
<td>Large</td>
<td>Simple</td>
<td>13.90</td>
<td>&gt; 24h</td>
<td>14.03</td>
</tr>
<tr>
<td></td>
<td>Any-path</td>
<td>32.61</td>
<td>&gt; 24h</td>
<td>33.01</td>
</tr>
<tr>
<td></td>
<td>ECMP</td>
<td>13.37</td>
<td>&gt; 24h</td>
<td>13.52</td>
</tr>
<tr>
<td></td>
<td>Ordered</td>
<td>249.48</td>
<td>&gt; 24h</td>
<td>1155.19</td>
</tr>
</tbody>
</table>

### Percentage of symbolic link costs

- **Different kinds of OSPF requirements**

- **Percentage of symbolic link costs**
Summary

- Learned about symbolic execution

- Learned about input synthesis for Datalog and how to use this for synthesizing network configurations

- Efficient OSPF synthesis using CEGIS
Next Lecture

So far: Analyze configurations for deterministic networks
- no probabilistic link failures
- no probabilistic routing

Problem: How can we derive guarantees for networks with probabilistic behaviors?
- How to calculate the probability of congestion in a network?
- How to check if traffic is correctly load-balanced along multiple links?

Next lecture: Probabilistic network analysis
References

Network-wide Configuration Synthesis, CAV 2017
Ahmed El-Hassany, Petar Tsankov, Laurent Vanbever, Martin Vechev
https://synet.ethz.ch

Network-wide Configuration Synthesis, NSDI 2018
Ahmed El-Hassany, Petar Tsankov, Laurent Vanbever, Martin Vechev
https://netcomplete.ethz.ch

Sketching Stencils, PLDI 2007
Armando Solar-Lezama, Gilad Arnold, Liviu Tancau, Rastislav Bodik, Vijay Saraswat, Sanjit Seshia