Structural Semantics Management: an Application of the Chase in Networking

Anduo Wang*, Mubashir Anwar#, Fangping Lan*, Matthew Caesar#
*Temple University #UIUC

***Abstract*—The value of database in advancing networking — in the paradigm shift from protocols to software-defined networking — was once highlighted by database-inspired management of network states. Moving beyond factual states, this paper considers *semantics management* a new frontier in the databases-networking knowledge "transfer", seeking to manage network policies via structural manipulation of the corresponding software (program). As a proof of concept, we make a case of semantics-based network transformation with the datalog structure and the chase, an elegant process for handling data dependencies (semantics). Our main result is an extension of the classic chase to *fauré-log*, a networking extension of datalog for the richer networking policies.**

***Index Terms*—Software-defined networking, network datalog, semantics-based transformation, the Chase**

I. Introduction

Database has played an active role in advancing networking research, notably, database inspired distributed network state management, a key enabler in the landscape changing movement of software-defined networks (SDN). Before custom built distributed key-value stores were available, production-scale SDN platforms achieved strong consistency among the replicas spanning across many datacenters by adopting the classical ACID notion and existing transactional databases [1]. As a second example, network verification, a topic that garnered wide interest and later borrows heavily from software engineering and formal methods, was first powered by datalog, which was lauded as a general modeling tool that enables declarative specification and fast simulation. With the clean and extensible network state management in datalog, it is not surprising that forerunners like Batfish [2] became a foundation (literally a component system) to many subsequent (often imperative and specialized) verifiers. If databases has helped shaping SDN, what is the next frontier?

One pain point in networking today is semantic management: As networks become more programmable (software-defined), the networks themselves are viewed as programs exhibiting richer semantics (policies), for which semantic management — maintaining the policy properties embedded in a network program — are pursued. Most tools for network semantics management take a primitive *behavioral approach* in which a network is modeled by a function whose inputs (packets) are exhaustively examined. For example, a network preserves its policy after an update if the tool cannot find a single input packet that exhibits the function — e.g., forwarding path for all packets — different. While great effort went into modeling the network function, the focus is to speed up evaluation on a huge input packet space. More advanced techniques capable of exploiting the network *structure* itself, however, is rare. The only *structural approach* we are aware is network transformers [3], [4] that use syntactic heuristics (e.g., based on network symmetry) to compress a network model into a smaller one while preserving certain properties.

In this paper, we consider semantic management a new frontier in network advancement by databases, pursuing the question: can we bring about structural network management in which the intended semantic management — analysis and transformation — intuitively maps into syntactic operation on the corresponding network representation? As a first step towards an affirmative answer, we study the concrete problem of semantics-based network transformation with the chase.

The chase [5], [6] is an elegant syntactic rewrite that takes a datalog query $\mathtt{Q}$ and a data dependency σ as input, transforms $\mathtt{Q}$ into $\mathtt{Q}'$ such that any "element" of $\mathtt{Q}$ that is incompatible with σ is intuitively corrected in $\mathtt{Q}'$ to satisfy σ, written as $\mathtt{chase}(\mathtt{Q}, \sigma) = \mathtt{Q}'$. To transform a network expressed in a datalog program $\mathtt{P}$, based on its policies given by a set of data constraints (i.e. dependencies) Σ, our idea is to repeatedly chase $\mathtt{P}$ with every dependency $\sigma \in \Sigma$, until we converge to a unique new network $\mathtt{P}'$ that properly reflects all policies.

The key is to extend the classic chase theory to networking. We first identified a limitation to the classic chase: the classic chase uses the standard query evaluation on a datalog program's instantiated database which is an incomplete database that requires more advanced evaluation. To address this mismatch, we generalize the chase to support richer semantics by developing a new algorithm $\mathtt{chase}(\mathtt{P}, \sigma)$, where both $\mathtt{P}, \sigma$ are datalog rules: by instantiating the $\mathtt{P}$ into an incomplete database instance $\mathtt{I}$, and processing σ as a data query over $\mathtt{I}$ by leveraging *fauré-log* evaluation, our earlier work on extending datalog to partial network state [8]. Our main finding is that, *the new chase with a set of dependencies remains Church-Rosser* [7] — the new chasing result remains unique (up to renaming of variable symbols) when terminating. While the classic chase transforms $\mathtt{P}$ with restricted dependencies into a single unique query, the new chase applicable to richer dependencies converts $\mathtt{P}$ into a unique *set* of programs.

II. A Running Example

We motivate semantics-based networking transformation by a running example in Figure 1: reachability between four groups of hosts (A,B,C,D in the left and right ovals) is controlled by the policies distributed at the 5 routers (center); $\mathtt{R}_2$ is configured to block any packet header with source matching B (i.e. prefix belonging to group B) and destination

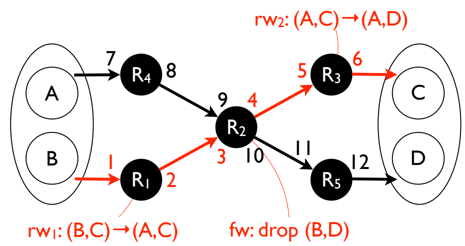

Fig. 1: Example reachability analysis in the presence of distributed policies: can host belonging in group B successfully send packets to hosts in D?

in D, $\mathtt{R_1}$ ($\mathtt{R_3}$, respectively) is set to rewrite header matching the pattern $(\mathtt{B},\mathtt{C})$ ($(\mathtt{A},\mathtt{C})$) to $(\mathtt{A},\mathtt{C})$ ($(\mathtt{A},\mathtt{D})$). That is, if the source of the header is in B (the destination of the header is in C), then modify the source (destination) to a host in A (D). In the presence of such rewrites, does $\mathtt{R_2}$, a node en-route all pair-wise paths, still enforce the semantics — preventing group B from contacting D? The answer is no. A host in B can reach a destination in D by injecting instead a packet with a header that matches $(\mathtt{B},\mathtt{C})$. Detecting such security hole with a existing behavioral analysis (e.g., Batfish [2]) requires insight into what packet to examine: the relevant packets include not only those with a header in $(\mathtt{B},\mathtt{D})$, but any packet created at group B.

Instead of improving behavioral analysis, we focus on the packet-manipulating network structure itself, that is a forwarding program P collectively driven by a set of policies Σ (=$\{\mathtt{rw_1},\mathtt{rw_2},\mathtt{fw}\}$) in Figure 1. While behavioral analysis partitions the packet space of P into the so called equivalent classes (ECs) [9], [10], so as to quickly and thoroughly exercise P's behavior (semantics) as governed by the policy set Σ, we ask, instead, how does Σ "modify" P structurally? And our goal is to syntactically transform program P, based on Σ, into a set of programs, such that each prescribes the network behavior (packet processing) on a particular EC in a more self-explanatory manner.

III. Datalog and the Classic Chase

To realize the semantics-based network transformation in § II, we present a first attempt with datalog and the classic chase. Datalog has long been accepted as an intuitive specification language for networking: the forwarding behavior along $\mathtt{R_1R_2R_3}$ naturally translates to r in Listing 1 where $\mathtt{F(flow, source, destination, location, next-hop)}$ is a predicate expressing that location (a switch interface in the network) forwards packet flow flow with header $\mathtt{(source, destination)}$ to the $\mathtt{next-hop}$. To transform the network to incorporate the constraint that the destination of a packet remains unchanged — simply a key dependency $\mathtt{k: flow \rightarrow destination}$, we only need to chase r with k, "correcting" the body of r — by the substitution $\mathtt{y_1/y_2, y_1/y_3, y_1/y_4, y_1/y_5, y_1/y_6, y_1/y}$ — to satisfy k.

```
r: R(x,y) :-F(f,x,y1,x,1), F(f,x2,y2,1,2), F(f,x3,y3,2,3),
     F(f,x4,y4,3,4), F(f,x5,y5,4,5), F(f,x6,y6,5,6),
     F(f,x7,y,6,y). % permitting header modifications along
     R1R2R3
/* the result of chasing r with k */
R(x,y1) :-F(f,x,y1,x,1),F(f,x2,y1,1,2),F(f,x3,y1,2,3),
     F(f,x4,y1,3,4), F(f,x5,y1,4,5), F(f,x6,y1,5,6),
     F(f,x7,y1,6,y1).
```

Listing 1: Example semantics-based network transformation

More generally, the classic chase is well-understood for data dependencies in the form of an equality generation dependency (*egd*) or tuple generation dependency (*tgd*). An example of *egd* is the key dependency k given by δ_1 in Listing 2, an example *tgd* is referential dependency (the presence of certain tuple in a relation implies the presence of another (probably in a different relation)). Both *tgd,egd* can be written as datalog rules if we allow (in)equality. This allows us to apply the chase to a datalog program q by a dependency σ by running σ on the "instantiation" of q (a symbolic database instance $\mathcal{D}$): *tgd* is just a regular datalog rule $\mathtt{h}:-\mathtt{b_1},\cdots,\mathtt{b_n}.$, the "evaluation" of which proceeds on $\mathcal{D}$ by adding the new atom h to $\mathcal{D}$ (program q); for an *egd* $\mathtt{e}:-\mathtt{b_1},\cdots,\mathtt{b_n}.$ (e is a substitution $\mathtt{y/y'}$ corresponding to an equality atom $\mathtt{y=y'}$ in δ_1), the evaluation is similar to egd except that instead of adding new goals to the rule, systematically applying the substitution.

```
δ1: y/y' :-F(f,x,y,u,w), F(f,x',y',u',w'). % datalog
     representation of k
/* datalog rules with (in)equality (involving constants)
     fails to evaluate on the symbolic database */
δ2: x/x', y/y' :-F(f,x,y,2,3), F(f,x',y',3,4), x≠1.2.3.4.
     % a firewall at R2 that filters source 1.2.3.4
δ3: x/x', y/y' :-F(f,x,y,2,3), F(f,x',y',3,4), ¬B(x). % a
     firewall at R2 that filters source from group B
```

Listing 2: Limitation of the classic chase

Unfortunately, the classic chase is too restricted for networking. The chase is hard to process even for a simple firewall policy for packets along $\mathtt{R_1R_2R_3}$ in Figure 1: δ_2 in Listing 2 specifies a firewall that allows packets to pass $\mathtt{R_2}$ only when its source is not 1.2.3.4, by involving the inequality with 1.2.3.4. δ_3 describes a firewall policy filtering any packets with a source from group B by using an auxiliary predicate B (not a database relation). We observe that the difficulty in chasing with policies given by such (general) datalog rules is that, these general constructs are not "evaluatable" on a symbolic database, because a symbolic database contains tuples with unknown values. For example, consider chasing r with δ_2, we have $\mathtt{F(f,x_3,y_3,2,3), F(f,x_3,y_3,3,4)}$ in the symbolic database $\mathcal{D}$, but we cannot determine whether $\mathtt{x_3} \neq 1.2.3.4$ holds or not, because $\mathtt{x_3}$ in $\mathcal{D}$ is a "symbolic" constant. Unlike a usual constant whose value we know, it is instantiated from a variable, with an uncertain value! For the same reason, when chasing with δ_3, we cannot decide the auxiliary predicate $\mathtt{B(x_3)}$.

IV. Extending the Chase to *Fauré*-log

Our goal is to develop a chase-like process to transform network behavior: given a network expressed in a datalog program p, we seek a network policy expression Σ, and a rewrite (chasing) process $\rightarrow_\Sigma$ (or abbreviated as $\rightarrow$ when Σ is clear), such that chasing p with Σ produces $\mathtt{p'}$ (written as $\mathtt{p} \rightarrow_\Sigma \mathtt{p'}$), where $\mathtt{p'}$ is a new program that properly incorporates the intention of Σ; the intention of Σ should be self-evident (not buried in a complex program) and the policy embedding rewrite $\rightarrow$ is self-explanatory (the modification to p reveals how the structure in p interacts with Σ).

```
r1: R(x,y1) :-F(f,x,y1,x,1),F(f,x,y1,1,2),
    F(f,x,y1,2,3),F(f,x,y1,3,4),F(f,x,y1,4,5),
    F(f,x,y1,5,6),F(f,x,y1,6,y1), [¬B(x),¬(A(x),C(y1))].
    % plain forwarding for prefixes not affected by any
    policies (e.g.,B(x) means x belongs to B)
r2: R(x,y6) :-F(f,x,y1,x,1),F(f,x,y1,1,2), F(f,x,y1,2,3),
    F(f,x,y1,3,4), F(f,x,y1,4,5), F(f,x,y6,5,6),
    F(f,x,y6,6,y6), [¬B(x),A(x),¬D(y1),C(y1),D(y6)]. %
    rewriting at R3 activated
r3: R(x,y6) :-F(f,x,y1,x,1),F(f,x2,y1,1,2),
    F(f,x2,y1,2,3),F(f,x2,y1,3,4),F(f,x2,y1,4,5),
    F(f,x2,y6,5,6),F(f,x2,y6,6,y6),
    [B(x),C(y1),A(x2),C(y1),D(y6)]. % rewriting at R1 and
    R3 activated
```

Listing 3: $r \rightarrow \{r_1, r_2, r_3\}$: the "combined effect" of the policies is clearly "pronounced" in the transformed program $\{\mathtt{r}_1, \mathtt{r}_2, \mathtt{r}_3\}$

For example, the network in Figure 1 (the forwarding behavior) is given by a 4-rule program (along 4 paths, namely $\mathtt{R}_1\mathtt{R}_2\mathtt{R}_3, \mathtt{R}_1\mathtt{R}_2\mathtt{R}_5, \mathtt{R}_4\mathtt{R}_2\mathtt{R}_3, \mathtt{R}_4\mathtt{R}_2\mathtt{R}_5$), each of which computes reachability along a specific path. Specifically, The rule r in Listing 1 specifies the network behavior along $\mathtt{R}_1\mathtt{R}_2\mathtt{R}_3$. We seek an expression for the three policies, such that chasing the program would embed those policies. Listing 3 illustrates the transformation result of r into three new rules, corresponding to the three equivalent classes determined by the policies. To achieve such network transformation, the rest of the section presents a design of Σ and a chase-like $\rightarrow$.

A. Extending the chase to symbolic network

Chasing a datalog program p with a dependency δ reduces to evaluating δ on the data instance $\mathcal{D}$ obtained from p, the challenge is that $\mathcal{D}$ is incomplete in the sense that the constant symbols instantiated from variables of p are not real constants, their values are unknown. To address this mismatch, we leverage incomplete databases research [8], [11]–[13]: based on our prior work *fauré-log*, a datalog extension for partial network information, we develop a dependency expression called *fauré*-dependency for networking policies, and extend the chase to *fauré*-dependency.

Specifically, we represent the symbolic instance during the chasing, which we call the symbolic network, by conditional tables (c-tables). C-tables allow both constants and variable symbols, while the constant has a "face value", the variable symbols denote unknown/uncertain values that are constrained by additional conditions (e.g., $\neg(\mathtt{x} = \mathtt{1.2.3.4})$ denotes an unknown value other than 1.2.3.4). Evaluation over such symbolic network is thus handled by *fauré-log* evaluation [8]: in a *fauré-log* program, the symbols include the usual constants and variables, and a new type of symbols called c-variables that are uncertain constants with additional conditions. The variables symbols now range over the domain of constants as well as the c-variables. The *fauré-log* evaluation enhances standard datalog evaluation by also properly manipulating the c-variable conditions. *Fauré*-dependency is just *fauré-log* rules with two exceptions (1) all the variable symbols are c-variables to capture the "uncertain constants" in a symbolic network, (2) in the head (left of the rule) we allow the chase actions (substitution and tuple generation), as shown in Listing 4. Intuitively, a *fauré-log* rule derives a symbolic head $\mathtt{H}(\mathtt{u})$ constrained by $\mathtt{C}(\mathtt{u})$ if the symbolic database contains $\mathtt{B}_1(\mathtt{u}_1), \cdots, \mathtt{B}_n(\mathtt{u}_n)$ under the condition $[\mathtt{C}(\mathtt{u}_1), \cdots, \mathtt{C}(\mathtt{u}_n)]$. That is, a *fauré*-dependency expresses *tgd* and *egd* conditionally.

```
/* network query on symbolic state */
H(u)[C(u)] :-B1(u1),···,Bn(un),[C(u1),···,C(un)]. %
    u,u1,...,un are tuples with constants and c-variables
/* network dependencies chasable on symbolic state */
H(u) :-B1(u1),···,Bn(un),[C(u1),···,C(un)]. % tgd: the
    presence of Bi's under the conditions Ci's implies H
[x/y, C(u)] :-B1(u1),···,Bn(un),[C(u1),···,C(un)]. % egd:
    substitute symbol x for y, C is a conjunction of
    (in)equality and auxiliary predicates
```

Listing 4: Query symbolic network by *fauré-log*, represent policies as *fauré*-dependency

Fauré-dependency can easily express all the network policies in Figure 1, as shown in Listing 5. For example, the header rewrite at $\mathtt{R}_1$ is given by σ_1, σ_2: σ_1 says that "irrelevant" packet headers (not matching the rewrite condition, captured by $\neg(\mathtt{B}(\mathtt{x}_1), \mathtt{C}(\mathtt{y}_1))$, where B, C are auxiliary predicates asserting group membership) in line 2) will pass $\mathtt{R}_1$ (through ingress interface 1 to egress 2) without change, thus we have the substitution in the head; on the other hand, σ_2 asserts that for packets matching the condition, the source address will be rewritten to a new address $\mathtt{x}_2$ in A. The header rewrite at $\mathtt{R}_3$ can be formulated similarly. The firewall at $\mathtt{R}_2$ is given by σ_3, σ_4: σ_3 describes the network behavior on packet not to be filtered, similar to σ_1; σ_4, for packets to be filtered, is interesting, it uses $\perp$ (falsehood), a special predicate (a 0-ary predicate always evaluating to false), in the head, implying a contradiction. Finally, σ_7 says that the packet header remains the same as long as it is at an interface not configured with a header rewrite or firewall.

```
1 /* rw1: rewriting policy at R1 */
2 σ1: [x1/x2, y1/y2] :-F(f,x1,y1,x1,1), F(f,x2,y2,1,2),
      [¬(B(x1),C(y1))]. % no action
3 σ2: [y1/y2, A(x2)] :-F(f,x1,y1,x1,1), F(f,x2,y2,1,2),
      [B(x1),C(y1)]. % rewrites source B→ A
4 /* fw: firewall at R2 */
5 σ3: [x1/x2, y1/y2] :-F(f,x1,y1,2,3), F(f,x2,y2,3,4),
      [¬(B(x1),D(y1))]. % no action
6 σ4: [⊥] :-F(f,x1,y1,2,3), F(f,x2,y2,3,4), [B(x1),D(y1)]. %
      filtering headers matching (B,D)
7 /* df (default policy): plain forwarding (no header
      modification) */
8 σ7: [x1/x2, y1/y2] :-F(f,x1,y1,_,u), F(f,x2,y2,u,_),
      [¬(u∈ {1,3,5,8,9,7,11})]. % when u does not match the
      location of any policy (rw1,rw2,fw)
```

Listing 5: Examples network policies (Figure 1) as *fauré-log*-dependencies

To chase with *fauré*-dependencies, we develop a new algorithm 1: Given a rule r, and a *fauré*-dependency σ, the intuition is, like the classic chase, to correct r — viewed as an symbolic instance — according to the requirement (substitution in egd, or the presence of new tuples in tgd) of σ. The main complexity is in handling the conditions: To decide the proper correction on the symbolic network state which are c-tables, we leverage the *fauré* evaluation engine to perform q(D) (line 3). When the result $\mathtt{H}'_\sigma[\psi_\sigma]$ is empty (line 4), the dependency is not "applicable" (e.g., the "premise" of the dependency is not satisfiable), so the chase halts; Otherwise, we proceed to compute and evaluate the new conditions under a systematic substitution (line 6): if the new condition is UNSAT (line 7), it signals an "impossible" network state, meaning that r and σ are incompatible; on the other hand, if the new condition is satisfiable, we apply the corrections by systematic substitutions (line 6) or new predicate insertions ($\mathtt{H}'_\sigma$ in line 8).

Algorithm 1: The chase with *fauré*-dependency

input : *fauré-log* rule $\mathtt{r} : \mathtt{H_r} : -\mathtt{B_r}[\phi_\mathtt{r}]$,
fauré-dependency $\sigma : \mathtt{H}_\sigma[\mathtt{x/y}, \psi_\sigma] : -\mathtt{B}_\sigma[\phi_\sigma]$
output: $\mathtt{r} \rightarrow_\sigma \mathtt{r}'$

1 instantiate $\mathtt{B_r}[\phi_\mathtt{r}]$ into c-tables D ;
2 let q be $\mathtt{H}_\sigma[\psi_\sigma] : -\mathtt{B}_\sigma[\phi_\sigma]$;
3 let $\mathtt{H}'_\sigma[\psi'_\sigma] = \mathtt{q}(\mathtt{D})$ by *fauré-log* evaluation ;
4 **if** $\mathtt{H}'_\sigma[\psi_\sigma]$ *is empty;*
5 **then** halt
6 **else**
7 let $\phi'_\mathtt{r} = \phi_\mathtt{r}\{\mathtt{x/y}\}, \phi'_\sigma = \phi_\sigma\{\mathtt{x/y}\}$;
8 **if** $\phi'_\mathtt{r} \wedge \phi'_\sigma \wedge \psi'_\sigma$ *is UNSAT* **then** halt;
9 **else** let $\mathtt{r}'$ be $\mathtt{H_r}\{\mathtt{x/y}\} : -\mathtt{B_r}\{\mathtt{x/y}\}, \mathtt{H}'_\sigma, [\phi'_\mathtt{r}, \phi'_\sigma, \psi'_\sigma]$
return $\mathtt{r}'$;
10 **end**
11 **end**

B. Discussion: chasing fauré-dependencies is Church-Rosser

Our main conjecture is that chasing with *fauré*-dependencies, despite being complete for a larger class of network dependencies via a more sophisticated procedure (Algorithm 1), remains "Church-Rosser". Given a set of policy dependencies Σ (multiple network policies), if chasing a *fauré-log* rule p (we chase a multi-rule program by independently chasing each rule) with Σ by repeatedly chasing with $\sigma \in\Sigma$ is terminating, the ordering in which the σ's are chosen is insignificant. That is, for any terminating sequence of dependencies from Σ, $\mathtt{p} \rightarrow_{...} \cdots \rightarrow_{\sigma_\mathtt{k}} \mathtt{p_k} \rightarrow_{...} \cdots \mathtt{p}'$, the end result $\mathtt{p}'$ is unique, and we write $\mathtt{p} \rightarrow_\Sigma \mathtt{p}'$. This is particularly appealing for reasoning about the joint effects of a set of distributed policies (Figure 1) since the "interaction" between them is insignificant.

We also point out an interesting twist with the new chase: Let a chase sequence of r by Σ be $\mathtt{s_1}, \cdots, \mathtt{s_k}, \cdots$, such that for each k, $\mathtt{s_k}$ is the result of applying some $\sigma \in\Sigma$ to $\mathtt{s_{k-1}}$ ($\mathtt{s_k}$ is the result of chasing $\mathtt{s_{k-1}}$ with σ). The sequence is terminal if it is finite and no dependency in Σ can be further applied to it. In such cases, the chase with Σ is terminating and the last element is called its result. With these notions, Church-Rosser for the classic chase is shown in Figure 2 (a): all (terminating) chasing sequences converge to a single rule $\mathtt{r}'$. The unique end result in the case of *fauré*-dependencies, however, becomes a set of rules γ $(= \{\mathtt{r_1}, \cdots, \mathtt{r_n}\})$: the chase sequences still converge to the unique γ, but the individual chase sequences can lead to different elements $\mathtt{r_k} \in \gamma$. In particular, each $\mathtt{r_k} \in \gamma$ represents the policy-based network behavior for a specific equivalent class.

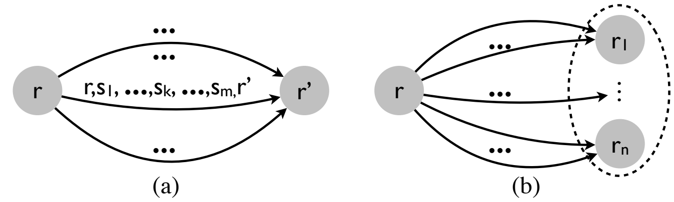

Fig. 2: Church-Rosser illustrated: (a) the classic chase; (b) the new chase with *fauré-log*.

Acknowledgments. This work was supported by National Science Foundation Award CNS-1909450, CNS-2145242.

References

[1] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: a distributed control platform for large-scale production networks," ser. OSDI'10, 2010.

[2] A. Fogel, S. Fung, L. Pedrosa, M. Walraed-Sullivan, R. Govindan, R. Mahajan, and T. Millstein, "A general approach to network configuration analysis," ser. NSDI'15. USA: USENIX Association, 2015.

[3] G. D. Plotkin, N. Bjørner, N. P. Lopes, A. Rybalchenko, and G. Varghese, "Scaling network verification using symmetry and surgery," in *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2837614.2837657

[4] R. Beckett, A. Gupta, R. Mahajan, and D. Walker, "Control plane compression," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 476–489. [Online]. Available: https://doi.org/10.1145/3230543.3230583

[5] A. Deutsch, A. Nash, and J. Remmel, "The chase revisited," in *Proceedings of the Twenty-Seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, ser. PODS '08. New York, NY, USA: Association for Computing Machinery, 2008. [Online]. Available: https://doi.org/10.1145/1376916.1376938

[6] D. Maier, A. O. Mendelzon, and Y. Sagiv, "Testing implications of data dependencies," *ACM Trans. Database Syst.*, vol. 4, no. 4, p. 455–469, dec 1979. [Online]. Available: https://doi.org/10.1145/320107.320115

[7] S. Abiteboul, R. Hull, and V. Vianu, Eds., *Foundations of Databases: The Logical Level*, Boston, MA, USA, 1995.

[8] F. Lan, B. Gui, and A. Wang, "Faure: a partial approach to network analysis," ser. ACM Workshop on Hot Topics in Networks (HotNets), November, 2021.

[9] P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, and S. Whyte, "Real time network policy checking using header space analysis," in *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, ser. nsdi'13. USA: USENIX Association, 2013, p. 99–112.

[10] P. Zhang, D. Wang, and A. Gember-Jacobson, "Symbolic router execution," in *Proceedings of the ACM SIGCOMM 2022 Conference*, ser. SIGCOMM '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 336–349. [Online]. Available: https://doi.org/10.1145/3544216.3544264

[11] R. van der Meyden, "Logical approaches to incomplete information: A survey," in *Logics for Databases and Information Systems (the book grow out of the Dagstuhl Seminar 9529: Role of Logics in Information Systems, 1995)*, J. Chomicki and G. Saake, Eds. Kluwer, 1998, pp. 307–356.

[12] T. Imieliński and W. Lipski, "Incomplete information in relational databases," *J. ACM*, vol. 31, no. 4, p. 761–791, Sep. 1984. [Online]. Available: https://doi.org/10.1145/1634.1886

[13] S. Abiteboul, P. Kanellakis, and G. Grahne, "On the representation and querying of sets of possible worlds," in *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '87. New York, NY, USA: Association for Computing Machinery, 1987, p. 34–48. [Online]. Available: https://doi.org/10.1145/38713.38724