# GUARDED NETKAT: SOUNDNESS, PARTIAL-COMPLETENESS, DECIDABILITY

A Thesis

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Science

by

Jacob Wasserstein

May 2023

**ABSTRACT**

The equational theory of Kleene Algebra with Tests (KAT) provides an algebraic framework to manipulate uninterpreted programs along with a PSPACE-complete procedure for deciding program equivalence. One of the most successful applications of the KAT theory is found in NetKAT, which instantiates the KAT metatheory to an interpreted language with primitives that support reasoning about packets in a network. NetKAT has found wide applications in reasoning about network correctness, with original applications to reachability, traffic isolation, and more. The power of KAT lies in its compact, expressive, and elegant signature - it describes a wide array of imperative programming paradigms, in particular ones which support nondeterminism at the level of programs.

Recent research has focused on the *guarded* fragment of KAT, known in the literature as Guarded Kleene Algebra with Tests (GKAT). This fragment of the KAT language trades expressivity for algorithmic performance. In particular, while remaining uninterpreted, guarded KAT programs have deterministic control flow: given a particular state the next action is completely determined, which differs from KAT in that the latter can compose programs nondeterministically. The upshot is that, even though GKAT can only encode a proper subset of valid KAT programs, the accompanying decision procedure for GKAT is nearly linear compared to the exponential worst-case in KAT. Furthermore, the loss of expressivity is mitigated by the fact that GKAT's control flow corresponds more closely with traditional programming constructs, such as if-then-else and while-loops.

Unfortunately, there have not yet been robust instantiations of the abstract GKAT theory in the same spirit as NetKAT for KAT. This is our present purpose: we aim to combine the primitive semantics and language model from NetKAT with the operations and metatheory from (abstract) GKAT. We offer a limited axiomatization for what we call *guarded NetKAT* (GNetKAT) and prove soundness and partial-completeness over this fragment. As we discover, it is deceptively difficult to adapt GKAT's uninterpreted theory to an interpreted setting, as the inclusion of extra axioms and commutativity conditions greatly complicates the metatheory. We proceed to develop a coalgebraic decision procedure to verify GNetKAT semantic program equivalence in nearly linear time, similar to but more elementary than that in the original GKAT setting. We conclude with discussion of applications in the vein of NetKAT. It is our hope that a proper understanding of how GKAT instantiates to its concrete models will motivate further developments in GKAT's theory, in particular towards resolving its non-algebraic axiomatization.

## BIOGRAPHICAL SKETCH

Jacob has been pursuing an M.S. in Computer Science from Cornell University since August of 2021. He attended Cornell University as an undergraduate in the College of Arts and Sciences from January 2018 to May 2021, where he studied mathematics and graduated *cum laude*. During his time as an undergraduate, Jacob's academic interests ranged from pure mathematics to programming to theoretical topics in computer science. His interest in formal verification was piqued in the spring of 2020 when he took CS 4160 - Formal Verification at Cornell, taught by Prof. Michael Clarkson. While originally planning to pursue a Ph.D. in probability theory, the time spent in Ithaca surrounded by accomplished theoreticians and the natural beauty of the Finger Lakes convinced him to remain at Cornell for his M.S. Jacob hopes to utilize the theoretical aspects of his studies to improve the quality and security of software written by users in the real world.

This document is dedicated to my mother and father, whose unconditional love and support for me has never faltered. I hope to honor them, and more importantly, I hope to make them proud. My accomplishments are theirs.

## ACKNOWLEDGEMENTS

My achievements would not be possible without my advisor, Alexandra Silva. She has consistently challenged me and encouraged me to expand my skills as a researcher, and she has always been reassuring and kind. She has offered me endless support, both academic and personal, without which I would have struggled greatly. She guided me in many ideas and proofs presented in this work, along with bestowing upon me a proper perspective on life, work, research, and how to balance each against the others. I am both proud, honored, and immensely grateful that Alexandra has stood by me through this process.

I would also like to thank the Department of Computer Science at Cornell for fostering an excellent environment to learn and grow, both as a student, researcher, and person. My time at Cornell, whether it be as an undergraduate or graduate student, has been consistently positive. The community and beauty found in Ithaca are unparalleled, and the Department, Gates Hall, and the field members in computer science and mathematics have always encouraged and embraced my curiosity. It has truly been a wonderful environment in which to live and to learn, one which I will look back upon fondly and wistfully.

## TABLE OF CONTENTS

CHAPTER 1

# INTRODUCTION

The field of formal methods has seen an explosion of new techniques for verifying equivalence of programs in recent years. The problem of deciding program equivalence has varied applications in static program analysis, compiler optimization, formal verification, and numerous domain-specific decision problems. A recurring example in this work is the algebraic framework of NetKAT [1], which allows one to encode network policies into a framework of regular expressions which comes ready-equipped with a PSPACE-complete decision procedure to decide equivalence of encoded policies. To do so, the authors draw on the theory of *Kleene Algebra with Tests* (KAT) introduced by Kozen in ([6], 1997), an equational system for reasoning about (uninterpreted) program equivalence, which itself is based upon the well-developed theory of *Kleene algebra* (KA). Much of the impetus to investigate Kleene algebra as an equational system was motivated by Kozen's 1994 finitary axiomatization and proof of completeness for Kleene algebras and the algebra of regular languages [5]. Kozen built upon this result with a proof of completeness and decidability of KAT, enabling researchers to encode many existing imperative paradigms into the KAT metatheory to leverage decidable program equivalence.

In order to adapt KAT to decide equivalence of network policies, the authors of the original NetKAT paper extended the KAT framework with domain-specific primitive actions, predicates, and axioms relating the new primitives to one another. In this way, they instantiated the uninterpreted KAT theory to a concrete setting, giving the language both a *packet history* semantics and a *trace semantics* according to certain regular languages of guarded strings. They

went on to prove the equivalence of these two semantics, along with completeness and decidability for the language with respect to these. The applications of their efforts (particularly in relating trace semantics to packet history semantics) include deciding connectivity among hosts in a network, deciding whether a network is loop-free, deciding whether a certain slice of a network isolates its traffic, and more. These applications illustrate the power of instantiating KAT's metatheory to different choices of actions, tests, and equations.

In a different direction, research published in 2019 described an efficient, expressive fragment of the KAT language known as *Guarded Kleene Algebra with Tests* (GKAT). More specifically, while KAT allows for nondeterministic programs with respect to its embedded Boolean subalgebra, GKAT restricts nondeterministic choice and iteration to their *guarded* variants, forcing all branch points and loops to test against some Boolean expression. While a seemingly innocuous change, the PSPACE-complete program equivalence procedure corresponding to KAT is improved to a nearly linear time equivalence procedure for GKAT programs. This tremendous speed-up is complemented by the fact that, while some expressivity is unavoidably lost by passing to the guarded variant of KAT, most "real-world" programs written in an imperative style are cleanly encoded as GKAT programs. As a result, large classes of existing imperative programs can be decided even more efficiently than in the original setting of KAT. The theoretical result which underpins the asymptotic speed-up relates to the sizes of automata (measured by size of state space) that are produced by the algorithm for equivalence checking. Indeed, automata and formal language theory underlie most of the semantic equivalence checking in these algorithms, with various perspectives on automata (e.g. universal coalgebra [7]) offering insight into the design and implementation of program equivalence procedures.

It should be emphasized that GKAT, like KAT, is an equational theory of *uninterpreted* programs, meaning that no assumptions about the semantics of primitive actions or predicates are made - in this way, the theories of both KAT and GKAT soundly overapproximate any consistent instantiation of the theories. However, it is still not a trivial matter to actually carry out the instantiation of either one of these theories to a specific choice of primitives, since primitive actions and tests often come with extra equations that one would wish to hold. Therefore, while most of the tools in the metatheory are present to be exploited, they usually cannot be applied in an "off-the-shelf" manner. Instead, as in the original NetKAT paper [1], more work needs to be done to relate concrete instantiatons of the theories to the theories themselves, especially when it comes to proving completeness and decidability of the extended theory.

We now arrive at the goal of this present work. While NetKAT has proven to be a very successful instantiation of KAT, there have not yet been many concrete instantiations of GKAT's metatheory. It is desirable to have such concrete implementations in order to demonstrate GKAT's applicability to verifying domain-specific programs, in addition to offering a working language with which to empirically document GKAT's theoretical efficiency gains. Therefore, we set out to present GNetKAT, which takes the primitive actions, tests, and equations from the NetKAT setting and adapts these to function within the more syntactically restrictive GKAT framework. Again, while the existing GKAT and KAT theories soundly overapproximate the semantics of any concrete instantiation, these are necessarily overapproximations. That is, whenever one finds a suitable choice of actions, tests, and equations for a novel language they wish to write, the individual must still perform non-trivial work to leverage the metatheory. There has been research into the area of automating this process of instantation - see [3].

For now, we proceed as those before us did, by manually specifying concrete syntax, semantics, and axioms for GNetKAT, proving soundness and a limited form of completeness for the semantics, then reducing to existing coalgebraic theory analogous to GKAT to obtain decidability. We go on to demonstrate several applications of GNetKAT in a similar sense to that of the original NetKAT paper.

The rest of this work is organized as follows. The remainder of Chapter 1 is devoted to the relevant preliminaries and notational conventions used throughout. Chapter 2 is devoted to presenting the syntax, semantics, and axioms for GNetKAT and *reduced* GNetKAT, which describes syntactically canonical representations of GNetKAT programs. Both a language model and a packet history model are presented for GNetKAT, and we prove the equivalence of these semantics towards potential applications. Chapter 3 investigates soundness and completeness of the GNetKAT axiomatization with respect to its language model, wherein we give elementary proofs of soundness (minus a fixpoint axiom) and loop-free completeness. Chapter 4 is dedicated to developing a coalgebraic semantic equivalence decision procedure similar to, but different than, that in the original GKAT setting. Finally, Chapter 5 integrates the results from previous chapters to describe applications of the GNetKAT framework in deciding certain properties of networks. We conclude after having described potential applications by comparing and contrasting with other novel research in the area of program equivalence.

## 1.1 Preliminaries

The following terminology and notation will be referenced throughout. Some basic familiarity with formal languages is assumed. The symbol $\triangleq$ is reserved for definitional equality, whereas = will be used more liberally - whether = is to be interpreted as program syntax or a logical symbol will always be clear from context. Given sets $X, Y$, the set of all functions $f : X \to Y$ is denoted $Y^X$. The set of subsets of a set $A$ will be interchangeably denoted $2^A$ or $\mathcal{P}(A)$ in light of the bijection $\{0,1\}^A \to \mathcal{P}(A)$. The *Iverson bracket* is defined on a proposition $\psi$ as $[\psi] = \begin{cases} 1 & \text{if } \psi \text{ holds} \\ 0 & \text{otherwise.} \end{cases}$ A *Boolean algebra* (BA) is a structure $(B, \vee, \wedge, 0, 1, \overline{(-)})$ satisfying

$$\begin{array}{llll} x \vee 0 = x & x \vee \overline{x} = 1 & x \vee y = y \vee x & x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z) \\ x \wedge 1 = x & x \wedge \overline{x} = 0 & x \wedge y = y \wedge x & x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z) \end{array}$$

Figure 1.1: Axioms of Boolean Algebra

A *Kleene algebra* (KA) is an idempotent semiring $(K, +, \cdot, 0, 1)$ equipped with a unary operation $(-)^* : K \to K$ satisfying the requirements below. Idempotency of the semiring, that $x + x = x$ for each $x \in K$, allows the definition of a partial order $\leq$ on $K$ by setting $x \leq y \iff x + y = y$. With this, the (universally-quantified) conditions for the star operation are:

$$1 + aa^* \leq a^* \qquad 1 + a^*a \leq a^*$$
$$ax \leq x \Rightarrow a^*x \leq x \quad xa \leq x \Rightarrow xa^* \leq x$$

The standard model of a Kleene algebra is the set of regular languages over

an alphabet $\Sigma$. That is, consider $(R(\Sigma), \cup, \cdot, \emptyset, \{\epsilon\}, (-)^*)$ where $R(\Sigma) \subseteq \mathcal{P}(\Sigma^*)$ is the smallest family of languages which contains $\{a\} \in R(\Sigma)$ for each $a \in \Sigma$ and is closed under union, concatenation, and Kleene star, defined for $L \in R(\Sigma)$ as $L^* \triangleq \bigcup_{n \in \mathbb{N}} L^n$ with $L^0 \triangleq \{\epsilon\}, L^{n+1} \triangleq L^n \cdot L$. This structure and its equational theory is also known as the *algebra of regular events*.

Given a Kleene algebra $(K, +, \cdot, 0, 1, (-)^*)$ and a Boolean algebra $(B, \vee, \wedge, 0, 1, \overline{(-)})$ which is a subalgebra of $K$, the two-sorted structure $(K, B, +, \cdot, 0, 1, (-)^*, \overline{(-)})$ is known as a *Kleene algebra with tests* (KAT). By subalgebra, we mean $B \subseteq K$ with the constants $0, 1 \in B$ and the equalities between functions $+|_B = \vee$ and $\cdot|_B = \wedge$ as maps $B \times B \to B$.

A central KAT of interest for us will be NetKAT as defined in [1]. In this language, there is an ambient collection of *fields* $f_1, \ldots, f_k$ with which one can assign and test against integer values. The Boolean primitive test $f = n$ intuitively does nothing if field $f$ is set to integer $n$, otherwise the program crashes. The primitive action $f \leftarrow n$ sets the field $f$ to have value $n$. The primitive action **dup** duplicates the current state of the fields in what is known as a *packet history*. The two-sorted syntax of NetKAT is given by

$$\begin{aligned} \mathit{Pred} \ni b, c &::= 0 \mid 1 \mid b + c \mid b \cdot c \mid f = n \mid \overline{b} \\ \mathit{Pol} \ni p, q &::= b \mid f \leftarrow n \mid \mathbf{dup} \mid p + q \mid p \cdot q \mid p^* \end{aligned}$$

where *Pred* is the embedded Boolean subalgebra of *Pol*. Intuitively, addition of policies $p + q$ is the nondeterministic choice between policies, multiplication of policies $p \cdot q$ is sequential composition of programs, and the star operation $p^*$ is nondeterministic iteration. To parse the syntax, we use the conventions that sequential composition binds tighter than choice and star binds tighter than

sequential composition, so the syntax $f \leftarrow n \cdot f' = m + q \cdot p^*$ should be parsed like $((f \leftarrow n) \cdot (f' = m)) + (q \cdot (p)^*)$. Furthermore, negation can only be applied to predicates, not arbitrary policies. We sometimes write $pq$ for $p \cdot q$ when brevity is needed.

The authors in [1] give an algebraic axiomatization for this language along with concrete semantics, and they go on to prove soundness and completeness for the axioms w.r.t the semantics. They conclude by giving a policy equivalence decision procedure and investigating potential applications - we aim to do much the same here. Before proceeding, though, we give some context on NetKAT and its intended functionality, in particular describing briefly how NetKAT models networks.

In NetKAT, there is an ambient, informal notion of a *network*, which consists of nodes (either switches on a router or an end-host; the distinction is not relevant at the node level) and links between nodes which represent valid connections between switches/hosts. A network in the NetKAT setting is then modeled using two NetKAT expressions: there is a network *policy*, which describes the individual behavior of nodes in the system, e.g. how they forward or drop packets, and a network *topology*, which describes the valid connections between nodes that can be used to send packets. Beyond this, the NetKAT framework is agnostic to whether nodes are "switches" or "hosts", and we will also ignore this distinction. Instead, in Chapter 5 we reuse the encoding of a network as two separate components, the policy (encoding *behavior* of the network) and the topology (encoding *structure* of the network). The particular details of networks or their encodings will not be addressed past this point, as the properties of the languages used to model programs/network policies is our primary interest.

Indeed, the network model is not mentioned in Chapters 2, 3, or 4.

An example of a NetKAT policy given in [1] is the following term which encodes the *topology* of a network:

$$\begin{aligned} t \;=\; & (sw = A \cdot pt = 2 \cdot sw \leftarrow B \cdot pt \leftarrow 1) + \\ & (sw = B \cdot pt = 1 \cdot sw \leftarrow A \cdot pt \leftarrow 2) + \\ & (sw = A \cdot pt = 1) + \\ & (sw = B \cdot pt = 2) \end{aligned}$$

where $sw$ and $pt$ are special fields representing the current location of the packet in the network. The term $t$ is the sum (union) of four terms; for example, the term $sw = A \cdot pt = 2 \cdot sw \leftarrow B \cdot pt \leftarrow 1$ intuitively asserts that the current switch is $A$ and then asserts that the current port is 2 (which just serves to identify a unique node in the network), then sets the current switch to be $B$ and sets the current port to be 1, which reflects that switch $A$ at port 2 forwards all of its traffic to switch $B$ at port 1. This term $t$ in its entirety encodes an internal link of the network between the switches $A$ and $B$ while modeling the links at the perimeter of the network with hosts 1 and 2 (which do not forward their traffic, hence have no outgoing edges). A network consisting of switches/hosts and edges between them is represented in NetKAT as a sum of uni-directional *links*, which encode the one-way edges of the directed graph corresponding to the network. Intuitively, a uni-directional link is just an edge in the directed graph of the network, describing how one node forwards traffic to another node. If two nodes can forward traffic to each other, there will be a uni-directional link in each direction. By taking the sum of links, we have nondeterministic choice between which path in the network that a given packet may follow.

Another example NetKAT term is the following which encodes forwarding of packets:

$$p \triangleq (dst = H_1 \cdot pt \leftarrow 1) + (dst = H_2 \cdot pt \leftarrow 2)$$

This policy $p$ is the nondeterministic choice, or *union*, of two smaller policies which forward packets destined for location $H_i$ to port $i$. The semantics for NetKAT is based on sets of histories of packets, in which the union of two policies on an input is encoded as the union of their outputs as sets. We adapt this semantics to GNetKAT in Section 2.3. For more examples of NetKAT policies of interest, see [1].

CHAPTER 2

# GNETKAT: SYNTAX, SEMANTICS, AXIOMS

We proceed to give the syntax, semantics, and axioms which underlie the GNetKAT framework. In this language, there is a finite list of *fields* $f_1, \ldots, f_k, k \geq 1$ which can be assigned non-negative integer values up to some upper bound $N \in \mathbb{N}$ with $N \geq 1$. These fields are interpreted as holding values in a packet, and the bounding $N$ for field values can be naturally interpreted as a level of desired bitvector precision. The language contains primitive actions and tests enabling assignment of integers to fields and tests of equality against constants, in addition to if-else, assert, and while-do constructs. These control flow structures allow to compose assignments and tests to form full-fledged network policies. We extend the base GKAT axioms with the packet algebra axioms as defined in the original NetKAT paper, along with providing a concrete language model as semantics. We go on to relate the language semantics with a packet history semantics in the same spirit as NetKAT.

Before proceeding, we describe an example GNetKAT program. Consider the imperative program in Figure 2.1. This portion of code is the imperative (deterministic) analog of the example NetKAT term in Section 1.1 (Preliminaries) which models the topology of a network: packets at certain locations are sent to other specific locations, with nodes at the perimeter of the network having no outgoing edges. While the NetKAT term in the previous section is described in terms of nondeterministic union of policies, we can model this using deterministic Boolean control flow: the cases $(sw = A \;\&\&\; pt = 2)$, $(sw = B \;\&\&\; pt = 1)$ $(sw = A \;\&\&\; pt = 1)$, and $(sw = B \;\&\&\; pt = 2)$ are disjoint, so it suffices to consider each case in isolation. We justify this formally later with the GNetKAT ax-

ioms, specifically the packet algebra axioms which imply $f = n \cdot \neg(f = m) \equiv f = n$ whenever $n \neq m$.

```
if (sw = A && pt = 2) {
    sw := B;
    pt := 1;
} else if (sw = B && pt = 1) {
    sw := A;
    pt := 2;
} else if (sw = A && pt = 1 || sw = B && pt = 2) {
    skip;
} else {
    assert false;
}
```

Figure 2.1: Encoding the topology from Section 1.1.

To encode this example program as a GNetKAT term, let $b_{Ai} \triangleq sw = A \cdot pt = i$ and $b_{Bi} \triangleq sw = B \cdot pt = i$ for $i \in \{1, 2\}$. Then the above imperative program corresponds in GNetKAT to

$$sw \leftarrow B \cdot pt \leftarrow 1 +_{b_{A2}}$$
$$sw \leftarrow A \cdot pt \leftarrow 2 +_{b_{B1}}$$
$$1 +_{b_{A1} \vee b_{B2}} 0$$

where the operator $e +_b g$, which intuitively represents **if** $b$ **then** $e$ **else** $b$, is right-associative. The term 1 is interpreted as **skip** or **assert true**, while 0 is interpreted as **assert false** or crashing. For another example, suppose we have a program $e$ and a special packet field *done* which is set to 0 at the start of execution, where our goal is to repeatedly run the program $e$ until it sets the *done* field to 1. The desired program is encoded in GNetKAT as $done \leftarrow 0 \cdot e^{(\neg(done=1))}$, which has the imperative reading "set *done* to 0; while *done* is not equal to 1,

execute $e''$. As one may start to glean, the syntax of GNetKAT closely mirrors the structure of general while-programs.

## 2.1 Syntax

We turn to presenting the syntax of the underlying language. The syntax of GNetKAT is given by

$$\begin{aligned} \textbf{BExp} \ni b, c &::= 0 \mid 1 \mid b + c \mid b \cdot c \mid f = n \mid \overline{b} \\ \textbf{GExp} \ni e, g &::= b \mid e \cdot g \mid e +_b g \mid e^{(b)} \mid f \leftarrow n \mid \textbf{dup} \end{aligned}$$

Intuitively, an element $b \in \textbf{BExp}$ asserts that $b$ is true in the current packet state, a product $e \cdot g$ performs $e$ followed by $g$, a guarded sum $e +_b g$ is **if** $b$ **then** $e$ **else** $g$, the primitives $f \leftarrow n$ assign the value $n$ to field $f$, the primitive **dup** records the current packet state in the current *history* of packets (more on this later), and a guarded loop $e^{(b)}$ is **while** $b$ **do** $e$. We note that $b + c$ for $b, c \in \textbf{BExp}$ should be interpreted as logical disjunction, *not* as the nondeterministic choice between two asserts - this latter interpretation is valid in NetKAT but not GNetKAT, since we no longer have nondeterministic choice at the command level. We will interchangeably write $b \vee c$ for $b + c$ whenever confusion may arise.

### 2.1.1 Complete Assignments and Tests

We also would like to define the *reduced* GNetKAT, which restricts GNetKAT programs to only those which contain *complete assignments* and *complete tests*. The reduced version of GNetKAT will be useful later when devising normal forms and proving partial-completeness. In this context, a complete assignment

is an expression of shape $f_1 \leftarrow n_1 \cdot \ldots \cdot f_k \leftarrow n_k$, i.e. a complete assignment is a term which updates each field name $f_i$ to be the value $n_i$. Likewise, complete tests are expressions of shape $f_1 = n_1 \cdot \ldots \cdot f_k = n_k$, i.e. a complete test is a conjunction of tests $f_i = n_i$ for each $i \in \{1, \ldots, k\}$. We let metavariables $\pi, \tau, \sigma$ range over complete assignments and $\alpha, \beta, \gamma$ range over complete tests. The set of complete assignments is denoted $\Pi$ and the set of complete tests is denoted At.

We note that both $\Pi$ and At are finite, since we only have finitely many fields $f_1, \ldots, f_k$ and each possible field value lies in the finite set $\{0, 1, \ldots, N\}$. Indeed, $\Pi$ and At are in bijection according to the integers $(n_1, \ldots, n_k)$, i.e.

$$(f_1 \leftarrow n_1 \cdot \ldots \cdot f_k \leftarrow n_k) \;\mapsto\; (f_1 = n_1 \cdot \ldots \cdot f_k = n_k)$$

defines a bijection from $\Pi$ to At. Given a $\pi \in \Pi$ we write $\alpha_\pi \in$ At for the corresponding complete test, and likewise if $\alpha \in$ At we write $\pi_\alpha \in \Pi$ for the corresponding complete assignment.

We often call $\alpha, \beta \in$ At *atoms*, since they are the minimal nonzero elements of the free Boolean algebra generated by primitive tests of shape $f = n$, where the ordering is entailment: $b \leq c \iff b \vee c \equiv_{\mathbf{BExp}} c$. This ordering is easily seen to be a partial order, and furthermore $b \leq \alpha \iff b \vee \alpha \equiv_{\mathbf{BExp}} \alpha \iff b \in \{0, \alpha\}$ which is justification for the terminology. These atoms play an essential role in providing semantics to GNetKAT terms (as they do in the original NetKAT [1]) as they represent a complete snapshot of the current state of the packet values.

One may wonder about the ordering of the primitive assignments (resp. tests) in a complete assignment (resp. test) - we will later introduce axioms which justify the convention of writing these terms in a fixed order according

to the ordering $f_1, \ldots, f_k$. That is, supposing there are only two fields $f_1$ and $f_2$, then $f_1 \leftarrow n_1 \cdot f_2 \leftarrow n_2$ is a complete assignment according to our definition above, but $f_2 \leftarrow n_2 \cdot f_1 \leftarrow n_1$ performs the assignments in the "wrong order". Our axiomatization in Section 2.3 will identify these expressions, allowing us to ignore the distinction up to provable equivalence.

### 2.1.2 Reduced Syntax

We are now ready to define the reduced GNetKAT syntax. The terms are

$$\begin{array}{rrcl} \textit{Complete Tests}: & \alpha, \beta & ::= & f_1 = n_1 \ldots f_k = n_k \\ \textit{Complete Assignments}: & \pi, \tau & ::= & f_1 \leftarrow n_1 \ldots f_k \leftarrow n_k \\ \textit{Sums of Atoms}: & A, B & ::= & \{\alpha_1, \ldots, \alpha_m\},\ m \geq 0 \\ \mathbf{GExp}^-: & e, g & ::= & \alpha \mid e \cdot g \mid e +_\alpha g \mid e^{(A)} \mid \pi \mid \mathbf{dup} \end{array}$$

There are a couple remarks that we should mention before proceeding. First, the set notation for sums of atoms is used to represent logical disjunction with no repeated terms, i.e. the intuitive interpretation of $A = \{\alpha_1, \ldots, \alpha_m\}$ for $m \geq 0$ is $\vee_{i=1}^m \alpha_i$ where $i \neq j \implies \alpha_i \neq \alpha_j$, with $m = 0$ corresponding to the empty disjunction: $\vee_{i=1}^0 \alpha_i = 0$. Now, the Boolean expression $\vee_{i=1}^m \alpha_i$ is (syntactically) dependent on the chosen ordering $\alpha_1, \ldots, \alpha_k$, but for any other ordering $\alpha_{i_1}, \ldots, \alpha_{i_k}$ it is provable from the Boolean algebra axioms that $\vee_{j=1}^m \alpha_j \equiv_{\mathbf{BExp}} \vee_{j=1}^m \alpha_{i_j}$. Since our notion of provable equivalence $\equiv$ defined in Section 2.3 will be a congruence for the GKAT operations that subsumes $\equiv_{\mathbf{BExp}}$, we are justified in using sets of atoms as a canonical notation for disjunction as long as we fix some global ordering on At, say the lexicographic ordering - it has no effect on the rest of the metatheory.

Secondly, one may wonder why we must introduce sums of atoms to the grammar at all, in particular why base asserts and guarded-choice may use atoms, but loops in general are forced to guard on sums of atoms. The reason for this is that an assertion of a sum of atoms, say $A = \{\alpha_1, \ldots, \alpha_m\}$, will end up being provably equivalent to the nested if-else $1 +_{\alpha_1} (1 +_{\alpha_2} (\ldots +_{\alpha_m} 0))$. Furthermore, we can exploit the skew-associativity axiom of GKAT (given in Section 2.3) to infer $e +_{\alpha \vee \beta} g \equiv e +_{\alpha} \Big(e +_{\beta} g\Big)$, so by induction any occurrence of $e +_A g$ can be replaced by $e +_{\alpha_1} (e +_{\alpha_2} (\ldots +_{\alpha_m} g))$. It is only for loops that we cannot be rid of sums of atoms in the guard, since in general there will be no way to solve a fixpoint equation to express $e^{(A)}$ in terms of $e^{(\alpha_1)}, \ldots, e^{(\alpha_m)}$.

Thirdly, the constants 0 and 1 are not explicitly given in the reduced syntax. However, they can be soundly encoded in the reduced language with respect to the semantics of the full language. This is done by defining $0 \triangleq e^{(\mathrm{At})}$ and $1 \triangleq e^{(\emptyset)}$ for some fixed expression $e$ and allowing for any expressions of shape $g^{(\mathrm{At})}$ or $h^{(\emptyset)}$ to be substituted for $e^{(\mathrm{At})}$ or $e^{(\emptyset)}$ respectively. The next section gives the semantics according to which this reasoning is sound.

## 2.2 Language Semantics

We now turn to giving semantics to both GNetKAT and reduced GNetKAT. The key semantic interpretation in this context is the *language model* for (reduced) GNetKAT, which interprets (reduced) GNetKAT expressions as *guarded languages* which consist of *guarded traces*, these being nonempty strings of the form $\alpha \cdot \pi_1 \cdot \mathbf{dup} \ldots \cdot \mathbf{dup} \cdot \pi_n \in \mathrm{GT} \triangleq \mathrm{At} \cdot (\Pi \cdot \mathbf{dup})^* \cdot \Pi$. Before defining the language model for (reduced) **GExp**, however, we need another definition. Let $p, q$

range over the set $(\Pi \cdot \mathbf{dup})^*$. For this setting, we define the *guarded concatenation* operator $\diamond : \mathrm{GT} \times \mathrm{GT} \to \mathrm{GT}$ on guarded traces as

$$\alpha \cdot p \cdot \pi \diamond \beta \cdot q \cdot \pi' \triangleq \begin{cases} \alpha \cdot p \cdot q \cdot \pi' & \text{if } \alpha_\pi = \beta \\ \text{undefined} & \text{otherwise} \end{cases}$$

Despite being a partial function on guarded traces, guarded concatenation lifts to a total function on guarded languages as $L_1 \diamond L_2 \triangleq \{x \diamond y : x \in L_1, y \in L_2\}$. Finally, if $B \subseteq \mathrm{At}$ then set $\overline{B} \triangleq \mathrm{At} \setminus B$.

We are now ready to define the language semantics for GNetKAT. The type of the semantics for GNetKAT programs $e \in \mathbf{GExp}$ is $[\![e]\!] \subseteq \mathrm{GT}$. We first start by giving the language model for reduced GNetKAT, then we proceed to give semantics for the full grammar. For $A \subseteq \mathrm{At}$ introduce the notation $[\![A]\!] \triangleq \{\alpha \cdot \pi_\alpha : \alpha \in A\}$. For terms in $\mathbf{GExp}^-$, the semantics is

$$\begin{array}{rcl@{\qquad}rcl} [\![\alpha]\!] & \triangleq & \{\alpha \cdot \pi_\alpha\} & [\![\mathbf{dup}]\!] & \triangleq & \{\alpha \cdot \pi_\alpha \cdot \mathbf{dup} \cdot \pi_\alpha : \alpha \in \mathrm{At}\} \\ [\![e \cdot g]\!] & \triangleq & [\![e]\!] \diamond [\![g]\!] & [\![e +_\alpha g]\!] & \triangleq & ([\![\alpha]\!] \diamond [\![e]\!]) \cup ([\![\mathrm{At} \setminus \{\alpha\}]\!] \diamond [\![g]\!]) \\ [\![e^{(A)}]\!] & \triangleq & (\bigcup_{n \in \mathbb{N}} ([\![A]\!] \diamond [\![e]\!])^{\underline{n}}) \diamond [\![\overline{A}]\!] & [\![\pi]\!] & \triangleq & \{\alpha \cdot \pi : \alpha \in \mathrm{At}\} \end{array}$$

We use the notation $L^{\underline{n}}$ for $L \subseteq \mathrm{At} \cdot (\Pi \cdot \mathbf{dup})^* \cdot \Pi$ to denote $L$ guarded-concatenated with itself $n$ times, formally defined by $L^{\underline{0}} \triangleq \{\alpha \cdot \pi_\alpha : \alpha \in \mathrm{At}\}$, $L^{\underline{n+1}} \triangleq L^{\underline{n}} \diamond L$. We can define the language semantics for the full grammar of GNetKAT as

$$
\begin{array}{rcl@{\qquad}rcl}
[\![b]\!] & \triangleq & \{\alpha \cdot \pi_\alpha : \alpha \leq b\} & [\![f \leftarrow n]\!] & \triangleq & \{\alpha \cdot \pi_\alpha[n/f] : \alpha \in \mathrm{At}\} \\
[\![e \cdot g]\!] & \triangleq & [\![e]\!] \diamond [\![g]\!] & [\![e +_b g]\!] & \triangleq & ([\![b]\!] \diamond [\![e]\!]) \cup ([\![\overline{b}]\!] \diamond [\![g]\!]) \\
[\![e^{(b)}]\!] & \triangleq & (\bigcup_{n \geq 0} ([\![b]\!] \diamond [\![e]\!])^n) \diamond [\![\overline{b}]\!] & [\![\mathbf{dup}]\!] & \triangleq & \{\alpha \cdot \pi_\alpha \cdot \mathbf{dup} \cdot \pi_\alpha : \alpha \in \mathrm{At}\}
\end{array}
$$

where, for $\pi \in \Pi$ and a field $f$ and integer $n$, $\pi[n/f]$ is the complete assignment where the assignment to $f$ is replaced with $f \leftarrow n$. As an example, the term $\pi \cdot \mathbf{dup} \cdot \pi'$ for $\pi \neq \pi'$ has

$$
\begin{aligned}
[\![\pi \cdot \mathbf{dup} \cdot \pi']\!] &\triangleq [\![\pi]\!] \diamond [\![\mathbf{dup}]\!] \diamond [\![\pi']\!] \\
&= \{\alpha \cdot \pi : \alpha \in \mathrm{At}\} \diamond \{\alpha \cdot \pi_\alpha \cdot \mathbf{dup} \cdot \pi_\alpha : \alpha \in \mathrm{At}\} \diamond \{\alpha \cdot \pi' : \alpha \in \mathrm{At}\} \\
&= \{\alpha \cdot \pi \cdot \mathbf{dup} \cdot \pi : \alpha \in \mathrm{At}\} \diamond \{\alpha \cdot \pi' : \alpha \in \mathrm{At}\} \\
&= \{\alpha \cdot \pi \cdot \mathbf{dup} \cdot \pi' : \alpha \in \mathrm{At}\}
\end{aligned}
$$

Consider another example term $f \leftarrow n +_{f=m} 1$ where $n \neq m$. Its language semantics is

$$
\begin{aligned}
[\![f \leftarrow n +_{f=m} 1]\!] &\triangleq [\![f = m]\!] \diamond [\![f \leftarrow n]\!] \cup [\![\neg(f = m)]\!] \diamond [\![1]\!] \\
&= \{\alpha \cdot \pi_\alpha : \alpha \leq f = m\} \diamond \{\alpha \cdot \pi_\alpha[f/n] : \alpha \in \mathrm{At}\} \cup \{\alpha \cdot \pi_\alpha : \alpha \leq \neg(f = m)\} \\
&= \{\alpha \cdot \pi_\alpha[f/n] : \alpha \leq f = m\} \cup \{\alpha \cdot \pi_\alpha : \alpha \leq \neg(f = m)\}
\end{aligned}
$$

Notice how in this previous example, there is no $\gamma \cdot \pi \in [\![f \leftarrow n +_{f=m} 1]\!]$ where $\pi$ sets $f \leftarrow m$. This is what one would expect, since the command $f \leftarrow n +_{f=m} 1$ is intuitively read as "if $f$ is equal to $m$, set $f$ to be $n$, otherwise do nothing".

The possible languages $L$ which arise as the semantics of some expression $e \in \mathbf{GExp}$ satisfy an imporant *determinancy property* analogous to (but stronger than) that from GKAT [11].

*Definition 2.1 (Determinacy property).* A language $L \subseteq \mathrm{GT}$ satisfies the determinacy property if for every $\alpha \in \mathrm{At}$ there exists at most one $y \in \Pi \cdot (\mathbf{dup} \cdot \Pi)^*$ for which $\alpha \cdot y \in L$. Languages with this property are said to be *deterministic*.

Note that an immediate consequence of this property is that $|L| \leq |\mathrm{At}|$ is finite. This notion of determinism is notably stronger than the definition given in abstract GKAT. In the abstract GKAT setting, primitive actions $p \in \Sigma$ are uninterpreted with respect to the Boolean subalgebra, meaning that $[\![p]\!] \triangleq \{\alpha p \beta : \alpha, \beta \in \mathrm{At}\}$, noting the difference in types: in the GKAT setting, guarded traces take the different shape $\mathrm{At} \cdot (\Sigma \cdot \mathrm{At})^*$. Intuitively, any (Boolean) state can be reached by any other (Boolean) state by performing $p$.

We call a GKAT language $L \subseteq \mathrm{At} \cdot (\Sigma \cdot \mathrm{At})^*$ *atom-deterministic* if for every pair $x, y \in L$, if $x$ and $y$ agree on their first $n$ atoms then they agree on their first $n$ actions (or lack thereof). For example, both $\{\alpha p \beta, \alpha p \gamma\}$ for $\beta \neq \gamma$ and $\{\alpha p \beta, \beta q \gamma\}$ are atom-deterministic, but $\{\alpha p \beta, \alpha\}$ and $\{\alpha p \beta, \alpha q \gamma\}$ for $p \neq q$ are not. The first example demonstrates an important part of the abstract GKAT theory: it allows for nondeterministic actions. Specifically, the control-flow of GKAT is required to be deterministic (where Booleans determine actions), but the primitive semantics (where actions determine Booleans) are not.

The notion of atom-determinism is just called determinism in the original GKAT setting [11], but the determinacy property satisfied by GNetKAT languages is strictly stronger (ignoring the difference in types momentarily), which is why we take care to distinguish the two. In particular, GNetKAT expressions $e$ denote a completely deterministic program. Its control flow is deterministic, and the manner in which its primitive actions influence state is also deterministic. This leads to the insight that, given a starting state $\alpha \in \mathrm{At}$, it has a unique

trace accepted by $e$ if there is one at all.

*Lemma 2.2.* Every expression $e \in \mathbf{GExp}$ satisfies $[\![e]\!] \subseteq \mathrm{GT}$ is deterministic.

*Proof.* By structural induction on $e$. The cases for $b$, $f \leftarrow n$, and **dup** are immediate.

For the case $e = g \cdot h$, by the IH it holds that for an arbitrary $\alpha \in \mathrm{At}$ there is at most one string $x$ which begins with $\alpha$ satisfying $x \in [\![g]\!]$. If this $x = \alpha \cdot y$ exists in $[\![g]\!]$, let $\pi_\beta$ be the trailing complete assignment in $y$. By the IH on $[\![h]\!]$, there exists at most one string beginning with $\beta$ in $[\![h]\!]$. If so, then there is exactly one string prefixed with $\alpha \in [\![g \cdot h]\!]$, 0 otherwise.

For the case $e = g +_b h$, let $\alpha \in \mathrm{At}$ and note that exactly one of $\alpha \leq b$ or $\alpha \leq \overline{b}$ holds. If $\alpha \leq b$, then there is no string prefixed with $\alpha \in [\![\overline{b}]\!] \diamond [\![h]\!]$, and there is at most one string prefixed with $\alpha \in [\![b]\!] \diamond [\![g]\!]$ by the IH on $g$, so the case holds. The other case $\alpha \leq \overline{b}$ is argued similarly.

For the case $e = g^{(b)}$, we prove by induction on $n \in \mathbb{N}$ that for all $\alpha \in \mathrm{At}$ there is at most one string prefixed by $\alpha$ in the language $([\![b]\!] \diamond [\![g]\!])^{\underline{n}}$. When $n = 0$, $([\![b]\!] \diamond [\![g]\!])^{\underline{0}} = \{\alpha \cdot \pi_\alpha : \alpha \in \mathrm{At}\}$ and the claim holds. When $n = k + 1$, let $\alpha \in \mathrm{At}$ and recall

$$([\![b]\!] \diamond [\![g]\!])^{\underline{k+1}} = [\![b]\!] \diamond [\![g]\!] \diamond ([\![b]\!] \diamond [\![g]\!])^{\underline{k}}$$

By the structural IH on $g$, there is at most one string prefixed with $\alpha \in [\![b]\!] \diamond [\![g]\!]$. If this string $x = \alpha \cdot y$ exists in $[\![b]\!] \diamond [\![g]\!]$, let $\pi_\beta$ be the trailing complete assignment in $y$. If $\beta$ is the prefix of some string $\beta \cdot z \in ([\![b]\!] \diamond [\![g]\!])^{\underline{k}}$ then by the IH

$z$ is unique. In this case there is exactly one string in $(\llbracket b \rrbracket \diamond \llbracket g \rrbracket)^{\underline{k+1}}$ prefixed with $\alpha$, so the case holds.

Therefore for all $\alpha \in \mathrm{At}$ and for all $n$ there exists at most one string in $(\llbracket b \rrbracket \diamond \llbracket g \rrbracket)^{\underline{n}}$. It follows then that for all $\alpha$ and $n$ there exists at most one string prefixed by $\alpha$ in $(\llbracket b \rrbracket \diamond \llbracket g \rrbracket)^{\underline{n}} \diamond \llbracket \overline{b} \rrbracket$.

It remains to show that if $x$ which begins with $\alpha$ satisfies both $x \in (\llbracket b \rrbracket \diamond \llbracket g \rrbracket)^{\underline{n}} \diamond \llbracket \overline{b} \rrbracket$ and $x \in (\llbracket b \rrbracket \diamond \llbracket g \rrbracket)^{\underline{m}} \diamond \llbracket \overline{b} \rrbracket$ then $n = m$. Suppose not and without loss of generality suppose $n < m$. By our claim let $x = y \diamond z \in (\llbracket b \rrbracket \diamond \llbracket g \rrbracket)^{\underline{n}} \diamond \llbracket \overline{b} \rrbracket$ be the unique factorization of $x$, i.e. $y \in (\llbracket b \rrbracket \diamond \llbracket g \rrbracket)^{\underline{n}}$ prefixed with $\alpha$ is unique and satisfies $x = y \diamond z = y \in (\llbracket b \rrbracket \diamond \llbracket g \rrbracket)^{\underline{n}}$. Since $m > n$ we know $m - n > 0$ and write

$$(\llbracket b \rrbracket \diamond \llbracket g \rrbracket)^{\underline{m}} = (\llbracket b \rrbracket \diamond \llbracket g \rrbracket)^{\underline{n}} \diamond (\llbracket b \rrbracket \diamond \llbracket g \rrbracket)^{\underline{m-n}}$$

Therefore we must be able to write $x \in (\llbracket b \rrbracket \diamond \llbracket g \rrbracket)^{\underline{m}}$ like $x = y \diamond w$ for $y \in (\llbracket b \rrbracket \diamond \llbracket g \rrbracket)^{\underline{n}}$, $w \in (\llbracket b \rrbracket \diamond \llbracket g \rrbracket)^{\underline{m-n}}$. Note that we intentionally reuse the variable $y$, since we know that it is unique and was assumed to exist earlier. We also know the trailing complete assignment $\pi_\beta$ in $y$ satisfies $\beta \leq \overline{b}$, since $y \diamond z$ exists for $z \in \llbracket \overline{b} \rrbracket$. Since $w \in (\llbracket b \rrbracket \diamond \llbracket g \rrbracket)^{\underline{m-n}}$ and $m - n > 0$, we know that the leading atom $\gamma$ of $w$ satisfies $\gamma \leq b$. Therefore $\beta \neq \gamma$ and we have reached a contradiction, for

$$x = y \diamond w = y' \cdot \pi_\beta \diamond \gamma \cdot w'$$

for some $y' \in \mathrm{At} \cdot (\mathbf{dup} \cdot \Pi)^*, w' \in \Pi \cdot (\mathbf{dup} \cdot \Pi)^*$ would be undefined.

We have exhausted all cases for $e$, so $\llbracket e \rrbracket$ is deterministic. □

## 2.3 Packet History Semantics

The language model mentioned previously is not the only semantics one can give to GNetKAT terms. The original NetKAT authors combined the understanding of language semantics with that of an operational *packet history* semantics. In this interpretation, a *packet* is a record of assignments $\{f_1 = v_1, \ldots, f_k = v_k\}$ describing the current values of the fields, and a *history* of packets is a nonempty list of packets. If $H$ is the set of all histories, then the packet history interpretation in NetKAT yields network policies (expressions) as functions $H \to \mathcal{P}(H)$. One reason that this type proved useful in the NetKAT setting is since NetKAT programs are often nondeterministic with respect to their embedded Boolean subalgebra. Even though our language is deterministic in this manner, we reuse the type $H \to \mathcal{P}(H)$ to cleanly capture the dropping of packets.

Formally, packet history semantics for GNetKAT is defined over **GExp** as having type $[\![e]\!]_H : H \to \mathcal{P}(H)$. It should be emphasized, though, that for each $e \in \mathbf{GExp}$ and $h \in H$, there will be *at most* one $h' \in [\![e]\!]_H(h)$, which may be seen as below. We define

$$\begin{array}{rcl|l} [\![f = n]\!]_H(pk :: l) & \triangleq & \begin{cases} \{pk :: l\} & pk.f = n \\ \emptyset & \text{otherwise} \end{cases} & pk ::= \{f_1 = v_1, \ldots, f_k = v_k\} \\ [\![b_1 \vee b_2]\!]_H(h) & \triangleq & [\![b_1]\!]_H(h) \cup [\![b_2]\!]_H(h) & h ::= pk :: l, \quad l ::= \langle\rangle \mid h \end{array}$$

$$
\begin{aligned}
[\![\overline{b}]\!]_H(h) &\triangleq \{h\} \setminus [\![b]\!]_H(h) & [\![f \leftarrow n]\!]_H(pk :: l) &\triangleq \{pk[f/n] :: l\} \\
[\![e \cdot g]\!]_H(h) &\triangleq ([\![g]\!]_H \ominus [\![e]\!]_H)(h) & [\![e +_b g]\!]_H(h) &\triangleq [\![b \cdot e]\!]_H(h) \cup [\![\overline{b} \cdot g]\!]_H(h) \\
[\![\mathbf{dup}]\!]_H(pk :: l) &\triangleq \{pk :: (pk :: l)\} & [\![e^{(b)}]\!]_H(h) &\triangleq \bigcup_{n \in \mathbb{N}} ([\![\overline{b}]\!]_H \ominus F^n)(h) \\
[\![0]\!]_H(h) &\triangleq \emptyset & [\![1]\!]_H(h) &\triangleq \{h\}
\end{aligned}
$$

where, for two functions $f, g : H \to \mathcal{P}(H)$ the function $g \ominus f : H \to \mathcal{P}(H)$ is their Kleisli composition defined by $(g \ominus f)(h) \triangleq \bigcup_{h' \in f(h)} g(h')$, and $F^n : H \to \mathcal{P}(H)$ for $n \in \mathbb{N}$ is defined by $F^0(h) \triangleq \{h\}, F^{n+1}(h) \triangleq ([\![b \cdot e]\!]_H \ominus F^n)(h)$. Note that we must explicitly give a definition for $[\![b_1 \vee b_2]\!]_H$ but not so for conjunction, since conjunction of tests is subsumed by the sequencing case. To interpret a set of atoms $A \subseteq \mathrm{At}$ as $[\![A]\!]_H$, we again treat $A$ as the disjunction of its atoms.

While the packet history semantics has a natural operational interpretation as relational semantics, the language model is much easier to work with in proofs. We would like to know that they are equivalent semantics, so that understanding of one yields insight into the other. This is captured in the following lemmas whose proofs are found in the appendix.

*Lemma 2.3.* For every reduced $e \in \mathbf{GExp}^-$, it holds that $[\![e]\!]_H(h) = \bigcup_{x \in [\![e]\!]} [\![x]\!]_H(h)$ for all $h$.

*Lemma 2.4.* For $x, y \in \mathrm{GT}$, $[\![x]\!]_H = [\![y]\!]_H$ if and only if $x = y$.

*Lemma 2.5.* For all $e, g \in \mathbf{GExp}^-$, $[\![e]\!]_H = [\![g]\!]_H$ holds if and only if $[\![e]\!] = [\![g]\!]$ holds.

Note that, by soundness (Theorem 3.3) and Lemma 3.6 (which states every $e \in \mathbf{GExp}$ is provably equivalent to some $e' \in \mathbf{GExp}^-$), two non-reduced programs $e, g \in \mathbf{GExp}$ satisfy $[\![e]\!] = [\![g]\!]$ if and only if $[\![e']\!] = [\![g']\!]$ if and only if $[\![e']\!]_H = [\![g']\!]_H$.

## 2.4 Axioms

We now turn to giving an axiomatization of GNetKAT and an analogue axiomatization for reduced GNetKAT. First, let $\equiv_{\mathbf{BExp}}$ be the smallest congruence (with respect to the Boolean operators) on **BExp** which contains the axioms of Boolean algebra as defined in Figure 1.1. With this relation, we can give the axiomatization for base GKAT along with the packet algebra axioms which specialize the theory to GNetKAT: the axioms are contained in Figure 2.2.

*Definition 2.6.* The relation $\equiv$ is the smallest congruence which respects the guarded choice, guarded loop, sequencing, and logical operations on tests which contains the axioms in Figure 2.2.

The axioms (U1.) - (U5.), (S1.) - (S5.), and (W1.), (W2.) are the base GKAT axioms as presented in [11], and axioms (P1.) - (P8.) are the packet algebra axioms as presented in [1]. In analogy with previously defined structures, a *guarded Kleene algebra with tests* is any two-sorted structure $(K, B, +, \cdot, 0, 1, \overline{(-)}, +_b, (-)^{(b)})$ where $(B, +, \cdot, 0, 1, \overline{(-)})$ is an embedded Boolean subalgebra of $K$ with maps $+_b : K \times K \to K$ and $(-)^{(b)} : K \to K$ for each $b \in B$ that satisfies the guarded union, guarded loop, and sequence axioms in Figure 2.2.

The sequence axioms are inherited from usual Kleene algebra and are intuitively understood in terms of sequential composition, skip (1) and failure (0). The guarded union axioms are intuitively understood in terms of conditional statements. For example, **if** $b$ **then** $e$ **else** $f$ should be equivalent to **if** $\overline{b}$ **then** $f$ **else** $e$, which is the content of the skew-commutativity axiom (U2.). The guarded loop axioms are slightly more complex - (W1.) encodes unrolling of while-loops and (W2.) tightening branching within while-loops.

**Guarded Union Axioms**

| | |
|---|---|
| U1. | $e +_b e \equiv e$ |
| U2. | $e +_b f \equiv f +_{\bar{b}} e$ |
| U3. | $(e +_b f) +_c g \equiv e +_{bc} (f +_c g)$ |
| U4. | $e +_b f \equiv be +_b f$ |
| U5. | $eg +_b fg \equiv (e +_b f) \cdot g$ |

**Guarded Loop Axioms**

| | |
|---|---|
| W1. | $e^{(b)} \equiv ee^{(b)} +_b 1$ |
| W2. | $(e +_c 1)^{(b)} \equiv (ce)^{(b)}$ |

All equivalences in $\equiv_{\mathbf{BExp}}$

**Sequence Axioms** (inherited from KA)

| | |
|---|---|
| S1. | $(e \cdot f) \cdot g \equiv e \cdot (f \cdot g)$ |
| S2. | $0 \cdot e \equiv 0$ |
| S3. | $e \cdot 0 \equiv 0$ |
| S4. | $1 \cdot e \equiv e$ |
| S5. | $e \cdot 1 \equiv 1$ |

**Packet Algebra Axioms**

| | |
|---|---|
| P1. | $f = n \cdot f' = n' \equiv f' = n' \cdot f = n$ if $f \neq f'$ |
| P2. | $f = n \cdot f = n' \equiv 0$ if $n \neq n'$ |
| P3. | $\sum_n f = n \equiv 1$ |
| P4. | $f \leftarrow n \cdot f \leftarrow n' \equiv f \leftarrow n'$ |
| P5. | $\mathbf{dup} \cdot f = n \equiv f = n \cdot \mathbf{dup}$ |
| P6. | $f = n \cdot f \leftarrow n \equiv f = n$ |
| P7. | $f \leftarrow n \cdot f = n \equiv f \leftarrow n$ |
| P8. | $f \leftarrow n \cdot f' = n' \equiv f' = n' \cdot f \leftarrow n$ if $f \neq f'$ |

Figure 2.2: GNetKAT Axioms

Notice that we do *not* include the unique fixpoint axiom (W3.) from the original GKAT setting, which says roughly that if $g \equiv eg +_b f$ then $g \equiv e^{(b)}f$ assuming that $e$ is *productive*. The side condition that $e$ is productive is essential to avoid unsound instantiations of the rule. Phrased differently, the unique fixpoint axiom (W3.) relies on a variant of Salomaa's *empty word property* in his 1966 axiomatization of Kleene Algebra [8]. While sound in the setting of uninterpreted actions, the empty word property becomes much harder to express correctly when more equations hold between actions and tests. We will return to this axiom and the reason for its exclusion in Section 3.1.

The axioms for packet algebra are intuitively understood in terms of assignment, with the extra caveats that **dup** commutes with all tests (P5.) and that each field $f$ must contain an integer value in $\{0, 1, \ldots, N\}$ (P3.). The analogous

packet algebra axioms for reduced GNetKAT are given in Figure 2.3.

$$\begin{array}{lll} \pi \equiv \pi \cdot \alpha_\pi & \alpha \cdot \mathbf{dup} \equiv \mathbf{dup} \cdot \alpha & \vee_{\alpha \in \mathrm{At}}\, \alpha \equiv 1 \\ \alpha \equiv \alpha \cdot \pi_\alpha & \pi \cdot \pi' \equiv \pi' & \alpha \cdot \beta \equiv 0 \text{ if } \alpha \neq \beta \end{array}$$

Figure 2.3: Reduced Packet Algebra Axioms

Based on the GKAT axioms alone, there are several useful facts following from the axioms which hold at the level of commands. We repeat the table shown in Section 3.1 of [11] in Figure 2.4, since many of these facts will be used later.

| | | | |
|---|---|---|---|
| (U3′.) | $e +_b (f +_c g) \equiv (e +_b f) +_{b+c} g$ | (W4.) | $e^{(b)} \equiv e^{(b)} \cdot \overline{b}$ |
| (U4′.) | $e +_b f \equiv e +_b \overline{b} \cdot f$ | (W4′.) | $e^{(b)} \equiv (be)^{(b)}$ |
| (U5′.) | $b \cdot (e +_c f) \equiv be +_c bf$ | (W5.) | $e^{(0)} \equiv 1$ |
| (U6.) | $e +_b 0 \equiv be$ | (W6.) | $e^{(1)} \equiv 0$ |
| (U7.) | $e +_0 f \equiv f$ | (W6′.) | $b^{(c)} \equiv \overline{c}$ |
| (U8.) | $b \cdot (e +_b f) \equiv be$ | (W7.) | $e^{(c)} \equiv e^{(bc)} \cdot e^{(c)}$ |

Figure 2.4: Derivable GKAT Facts

*Example 2.7.* As an illustration of these axioms, let us show $f \leftarrow n \cdot (e +_{f=n'} g) \equiv f \leftarrow n \cdot g$ assuming $n \neq n'$. Before proving this, we first show that $f = n \cdot \neg(f = n') \equiv f = n$. For proof of this, recall the axiom $f = n \cdot f = n' \equiv 0$ and that sequencing restricted to tests is logical conjunction (and is therefore commutative on tests). Since $\equiv$ is a congruence with respect to logical operations, add $f = n \cdot \neg(f = n')$ to both sides of $f = n \cdot f = n' \equiv 0$ to get (by purely Boolean reasoning)

$$
\begin{aligned}
f = n \cdot \neg(f = n') &\equiv f = n \cdot f = n' + f = n \cdot \neg(f = n') \\
&\equiv f = n \cdot (f = n' + \neg(f = n')) \\
&\equiv f = n \cdot 1 \\
&\equiv f = n
\end{aligned}
$$

We now return to showing that $f \leftarrow n \cdot (e +_{f=n'} g) \equiv f \leftarrow n \cdot g$. Consider

$$
\begin{aligned}
f \leftarrow n \cdot (e +_{f=n'} g) &\equiv f \leftarrow n \cdot f = n \cdot (e +_{f=n'} g) && \text{(P7.)} \\
&\equiv f \leftarrow n \cdot f = n \cdot (f = n' \cdot e +_{f=n'} g) && \text{(U4.)} \\
&\equiv f \leftarrow n \cdot (f = n \cdot f = n' \cdot e +_{f=n'} f = n \cdot g) && \text{(U5′.)} \\
&\equiv f \leftarrow n \cdot (0 \cdot e +_{f=n'} f = n \cdot g) && \text{(P2.)} \\
&\equiv f \leftarrow n \cdot (f = n \cdot g +_{\neg(f=n')} 0) && \text{(U2.) (S2.)} \\
&\equiv f \leftarrow n \cdot \neg(f = n') \cdot f = n \cdot g && \text{(U6.)} \\
&\equiv f \leftarrow n \cdot f = n \cdot g && (*) \\
&\equiv f \leftarrow n \cdot g && \text{(P7.)}
\end{aligned}
$$

where $(*)$ uses $\neg(f = n') \cdot f = n \equiv_{\mathbf{BExp}} f = n \cdot \neg(f = n')$ and our proved identity $f = n \cdot \neg(f = n') \equiv f = n$. □

CHAPTER 3

# SOUNDNESS AND PARTIAL COMPLETENESS

We would now like to address the issue of soundness and completeness of GNetKAT with respect to the language model introduced in the previous section. First, we begin this chapter by describing some essential differences between GNetKAT, abstract GKAT, and a variant of GKAT known as *skip-free* GKAT [10]. The latter two are theories of uninterpreted actions, while the former two have many syntactic similarities. Both abstract GKAT and skip-free GKAT have served as potential starting points for this work - we discuss the advantages and disadvantages associated with each. Next, we relate our semantics to that of both NetKAT and (abstract) GKAT. This will make clear which soundness-related facts need to be re-proven and which facts may be taken for granted based on prior work. After these relationships are understood, we progress to give the necessary tools to argue soundness followed by an elided proof of soundness. Finally, we conclude by devising a normal form for the loop-free fragment of **GExp** and proving completeness over this fragment.

## 3.1 GNetKAT, GKAT, Skip-Free GKAT

In devising a guarded variant of the existing NetKAT framework, one question needed to be addressed early on: which guarded KAT fragment should we work with? Abstract GKAT (c. 2019) was shown sound and complete with respect to its canonical language model and was developed before skip-free GKAT (c. 2023), with the former serving as an impetus for the latter. Specifically, skip-free GKAT, which is obtained from GKAT by eliminating skip and

all skip-like expressions, was developed to address two major drawbacks to the GKAT metatheory. These are:

(i) the GKAT axioms are not *algebraic*, meaning that they are not sound under algebraic substitution of arbitrary expressions. This means that, given $e \equiv e'$ which both mention an action symbol $p$, it is *not* true for all expressions $e''$ that $e[e''/p] \equiv e'[e''/p]$;

(ii) the GKAT completeness proof assumes an extra *uniqueness axiom* scheme asserting the uniqueness of solutions to certain left-affine systems of equations. This is essentially a generalized fixpoint axiom analogous to (W3.) in [11], but the authors mention that the generalized fixpoint rule does not seem to follow from (W3.) directly - they conjecture that it must be assumed.

Whether these two issues can be rectified for abstract GKAT remains open [11]. As a first-step towards addressing these issues, skip-free GKAT circumvents (i) and (ii) by restricting the syntax of abstract GKAT one step further. Specifically, skip-free GKAT is no longer a two-sorted structure, as Booleans $b \in \mathbf{BExp}$ are no longer command-level asserts. Instead, they are only present in guarded-choice $e +_b f$ and loops $e^{(b)}f$, where now loops must be followed by a sequenced command $f$ (to avoid $e^{(0)} = 1$). The only "Boolean" at the command level is 0 which represents crashing (failure). The authors in [10] give a sound and complete axiomatization of skip-free GKAT over its language model which is *algebraic* without assuming a uniqueness axiom scheme like the one from base GKAT. They do so by first reducing completeness of skip-free GKAT modulo bisimulation to the completeness of the theory of 1-free star expressions modulo bisimulation

developed by Grabmayer and Fokkink [2], then reducing completeness of skip-free GKAT w.r.t. language semantics to completeness of skip-free GKAT modulo bisimulation.

While skip-free GKAT has two major theoretical advantages over GKAT as described above, it is also significantly more difficult to actually write axioms for a specific choice of instantiated theory. In fact, skip-free GKAT was originally the targeted framework at the beginning of this writing, but certain persistent issues caused the author to shift focus. For example, suppose $\alpha_1, \ldots, \alpha_K$ enumerates At and consider the skip-free expression

$$e \triangleq \pi_{\alpha_1} +_{\alpha_1} (\ldots (\pi_{\alpha_K} +_{\alpha_K} 0))$$

In Lemma 3.5, we will show that in GNetKAT (which follows GKAT and *not* skip-free GKAT) the equivalence $e \equiv 1$ is derivable, which implies $e \cdot g \equiv g \equiv g \cdot e$ for any $g$ in our axiomatization. Therefore, even though $e$ is a valid skip-free program, it encodes the semantics of skip in a skip-free term. This alone is not a contradiction, but it does hint towards difficulties in reasoning axiomatically in the skip-free setting, specifically in how to avoid the equivalence $e \equiv 1$ being provable. For example, consider the packet algebra axiom $f = n \cdot f \leftarrow n \equiv f = n$ of NetKAT and GNetKAT. This is *not* a valid skip-free GKAT axiom due to the sequencing of a test and an action. Instead, we could try something like this in the skip-free setting:

$$f \leftarrow n \cdot (e +_{f=n \wedge b} g) \equiv f \leftarrow n \cdot (e +_b g)$$

This is a reasonable axiom to add and allows to derive equivalences like $f \leftarrow n \cdot (e +_{f=n} g) \equiv f \leftarrow n \cdot e$ in the skip-free setting. Unfortunately, this axiom does not allow us to derive the analog of guardedness, which would be $e +_{f=n} g \equiv (f \leftarrow n \cdot e) +_{f=n} g$. Even worse, return for a moment to $f \leftarrow n \cdot (e +_b g)$. Intuitively, if $b$ does not mention $f$ then $f \leftarrow n \cdot (e +_b g)$ should be equivalent to $(f \leftarrow n \cdot e) +_b (f \leftarrow n \cdot g)$. This equivalence is simple enough to derive in full GKAT due to the presence of commutativity conditions $f \leftarrow n \cdot f' = m \equiv f' = m \cdot f \leftarrow n$ between actions and tests. However, these commutativity conditions cannot be expressed directly in skip-free GKAT, instead necessitating more axioms which awkwardly relate the behavior of tests in control-flow structures to that of primitive actions. This seems to be a recurring phenomenon, especially when one is interested in concrete theories with several equations relating actions and tests. Whenever one wishes to describe the effects of an action on a test or vice versa, one needs to devise a way to relate the two through skip-free GKAT's restrictive syntax. This is where the lack of a two-sorted structure is most relevant - the accompanying theory is more difficult to use. It is for this reason that the author decided to adapt NetKAT to base GKAT instead of skip-free GKAT, as a major impetus for this writing was precisely to instantiate the guarded KAT metatheory to concrete actions, tests, and equations.

This is not to say that the GKAT theory carries over cleanly into GNetKAT either, though. In particular, naively combining NetKAT's language model and GKAT's axiomatization leads to soundness issues, particularly for GKAT's fixpoint axiom (W3.). To state this axiom in the abstract GKAT setting, let $p \in \Sigma$ be an uninterpreted action (i.e. element of finite alphabet) and $G, B$ the set of abstract GKAT expressions and Booleans over $\Sigma$ respectively and define $E : G \to B$ inductively by

$$E(b) \triangleq b \qquad E(e^{(b)}) \triangleq \overline{b} \qquad E(p) \triangleq 0$$
$$E(e \cdot g) \triangleq E(e) \cdot E(g) \qquad E(e +_b g) \triangleq b \cdot E(e) + \overline{b} \cdot E(g)$$

This defines the empty word property for $e \in G$, where $E(e) \equiv 0$ if and only if the language interpretation $[\![e]\!] \subseteq \mathrm{At} \cdot (\Sigma \cdot \mathrm{At})^*$ for $e \in G$ does not contain just the string $\alpha$ for any $\alpha \in \mathrm{At}$. Then, in abstract GKAT the axiom (W3.) says that $E(e) \equiv_{\mathbf{BExp}} 0$ and $g \equiv eg +_b f$ implies $g \equiv e^{(b)} f$. This is sound for GKAT's language model, but unfortunately, in the GNetKAT setting, naively setting $E(f \leftarrow n) \triangleq 0$ leads to the analog of (W3.) being unsound. This is due, in part, to a difference in types: GKAT languages are subsets of $\mathrm{At} \cdot (\Sigma \cdot \mathrm{At})^*$, whereas GNetKAT languages are subsets of $\mathrm{At} \cdot \Pi \cdot (\mathbf{dup} \cdot \Pi)^*$. In particular, every GNetKAT trace starts with an $\alpha \in \mathrm{At}$ and ends with a $\pi \in \Pi$, so it is not clear how one would detect, syntactically or semantically, when "no action" is performed on a given start state $\alpha$.

Specifically, suppose we included axiom (W3.) from GKAT in GNetKAT, along with a modified $E : \mathbf{GExp} \to \mathbf{BExp}$ defined in the obvious way with $E(f \leftarrow n) \triangleq 0$ (ignoring $E(\mathbf{dup})$ for now). Observe then that $\pi_\alpha \equiv \pi_\alpha \cdot \pi_\alpha$ from the packet algebra axioms, so $\pi_\alpha \equiv \pi_\alpha \pi_\alpha +_\alpha \pi_\alpha$ by idempotence. Since $E(\pi_\alpha) = 0$, applying the fixpoint axiom yields

$$\pi_\alpha \equiv \pi_\alpha \pi_\alpha +_\alpha \pi_\alpha \wedge E(\pi_\alpha) \equiv 0 \implies \pi_\alpha \equiv (\pi_\alpha)^{(\alpha)} \pi_\alpha$$

But the equation $[\![\pi_\alpha]\!] = [\![(\pi_\alpha)^{(\alpha)} \pi_\alpha]\!]$ does not hold as subsets of $\mathrm{At} \cdot \Pi \cdot (\mathbf{dup} \cdot \Pi)^*$: the set $[\![\pi_\alpha]\!]$ has a trace prefixed with $\alpha$, while $[\![(\pi_\alpha)^{(\alpha)} \pi_\alpha]\!]$ does not. Furthermore,

slight modifications to $E$, like setting $E(f \leftarrow n) \triangleq f = n$, do not seem to fix the issue. Instead, the empty word property is a subtle, semantic concept, one which is significantly easier to formulate when actions are opaque (uninterpreted). In the presence of packet algebra axioms like $\alpha \equiv \alpha \cdot \pi_\alpha$ and the type of traces $\mathrm{At} \cdot \Pi \cdot (\mathbf{dup} \cdot \Pi)^*$ where a trace always ends in a $\pi \in \Pi$, it is not clear what form the empty word property would even take - does it describe accepting some trace of the form $\alpha \cdot \pi_\alpha$? Does it describe accepting *every* trace of the form $\alpha \cdot \pi_\alpha$? A precise formulation of "productivity" in the presence of commutativity conditions needs to be tacked down before the fixpoint axiom can be formulated in this setting.

This point serves to highlight some of the difficulties in instantiating the abstract GKAT framework. Its corresponding equational theory was only shown sound and complete with respect to its canonical (free) language model. Due to the non-algebraic side condition for (W3.), extreme care must be taken when designing concrete instantiations of GKAT to not break soundness. It is for this reason that we exclude (W3.) entirely from our axiomatization, only proving soundness and completeness over a subset of the GKAT axioms.

## 3.2 Soundness

To proceed with proof of soundness of the GNetKAT axioms against its language model, recall that the GNetKAT language model interprets a GNetKAT expression $e \in \mathbf{GExp}$ as $[\![e]\!] \subseteq \mathrm{GT}$ a set of guarded traces $\alpha \cdot \pi_1 \cdot \mathbf{dup} \cdot \ldots \cdot \mathbf{dup} \cdot \pi_n$ ($n \geq 1$) which describe valid executions of $e$. For reduced GNetKAT, this semantics is exactly the reduced language model $G$ defined in Figure 7, Sec-

tion 4.2 of the original NetKAT paper [1] when restricted to guarded expressions. That is, let $\phi$ denote the homomorphism from reduced GNetKAT to reduced NetKAT arising from setting $\phi(e +_\alpha g) \triangleq \alpha \cdot \phi(e) + \left(\sum_{\beta \neq \alpha} \beta\right) \cdot \phi(g)$ and $\phi(e^{(A)}) \triangleq ((\sum_{\alpha \in A} \alpha) \cdot \phi(e))^* \cdot \sum_{\alpha \in \overline{A}} \alpha$ where $\phi$ acts as the identity on actions and tests. Then $G \circ \phi = [\![\cdot]\!]$ as maps $\mathbf{GExp}^- \to 2^{\mathrm{GT}}$, which follows directly from the definitions of $G, \phi$, and $[\![\cdot]\!]$. Indeed, if we define $\tilde{G}$ from $G$ to operate on all of NetKAT by setting $\tilde{G}(f \leftarrow n) \triangleq \{\alpha \cdot \pi_\alpha[f/n] : \alpha \in \mathrm{At}\}$ and $\tilde{G}(b) \triangleq \{\alpha \cdot \pi_\alpha : \alpha \leq b\}$, then with $\tilde{\phi}(e +_b g) \triangleq b \cdot e + \overline{b} \cdot g$ and $\tilde{\phi}(e^{(b)}) \triangleq (b \cdot e)^* \cdot \overline{b}$, the relationship is expressed as $\tilde{G} \circ \tilde{\phi} = [\![\cdot]\!]$ where these are maps $\mathbf{GExp} \to 2^{\mathrm{GT}}$. The payoff of this is that the soundness of the packet algebra axioms may be taken for granted. This is because the packet algebra axioms were already shown to be sound with respect to the relational packet history model in the original NetKAT setting, which was proven isomorphic to the extension $\tilde{G}$. Therefore, we need only reprove soundness for the rest of the axiomatization (which comes from GKAT).

Unfortunately, each GKAT axiom was verified sound in [11] with respect to a different language model than the one presented here. Specifically, an (abstract) GKAT term $e$ is interpreted as $[\![e]\!] \subseteq \mathrm{At} \cdot (\Sigma \cdot \mathrm{At})^*$, where now $\Sigma$ is a (finite) set of primitive actions and At is still the minimal elements of the free Boolean algebra generated by primitive tests $t \in T$ ($T$ finite). In particular, since GKAT makes no assumptions about the semantics of its primitives, it soundly overapproximates any instantiation by setting $[\![p]\!] \triangleq \{\alpha \cdot p \cdot \beta : \alpha, \beta \in \mathrm{At}\}$ for $p \in \Sigma$. In essence, beginning in any state $\alpha$ and for any desired end state $\beta$, executing $p$ takes one from $\alpha$ to $\beta$. This is markedly different than the semantic interpretation for GNetKAT, as one would expect for a nontrivial instantiation of the theory - it is *not* true in GNetKAT that, given any two complete assignments $\alpha, \beta$ and action $f \leftarrow n$, executing $f \leftarrow n$ in state $\alpha$ results in state $\beta$. It is in this sense that abstract GKAT

soundly overapproximates any instantiation, with the inevitable conclusion that we must re-prove the soundness of the abstract GKAT axioms.

### 3.2.1 Guarded Concatenation

To prove soundness of the GKAT axioms with respect to the GNetKAT language model, we need to leverage some basic facts about the guarded concatenation operator

$$\alpha \cdot x \cdot \pi \diamond \beta \cdot y \cdot \pi' \triangleq \begin{cases} \alpha \cdot x \cdot y \cdot \pi' & \text{if } \alpha_\pi = \beta \\ \text{undefined} & \text{otherwise} \end{cases}$$

and its lifting to languages.

*Lemma 3.1.* The guarded concatenation operator on languages $\diamond : 2^{\mathrm{GT}} \times 2^{\mathrm{GT}} \to 2^{\mathrm{GT}}$ satisfies the following properties:

(a) $\diamond$ left and right-distributes over union;

(b) $\diamond$ is associative;

(c) The set $I \triangleq \{\alpha \cdot \pi_\alpha : \alpha \in \mathrm{At}\}$ satisfies $I \diamond K = K = K \diamond I$ for all $K \subseteq \mathrm{GT}$.

Parts (b) and (c) of Lemma 3.1 show that $(2^{\mathrm{GT}}, \diamond)$ is a monoid with identity element $I$. The lemma in its entirety demonstrates $(2^{\mathrm{GT}}, \cup, \diamond, \emptyset, I)$ is an idempotent semiring. The proof of Lemma 3.1 is found in the appendix.

### 3.2.2 Main Result

*Remark 3.2.* Given $b, c \in \mathbf{BExp}$, it is readily seen that $[\![b]\!] \diamond [\![c]\!] = [\![bc]\!] = [\![cb]\!] = [\![c]\!] \diamond [\![b]\!]$. In fact, more is true: for $b, c \in \mathbf{BExp}$ it holds that $b \equiv_{\mathbf{BExp}} c$ if and only if $[\![b]\!] = [\![c]\!]$. This is soundness and completeness with respect to the Boolean algebra axioms on **BExp**, which follows by definition of $[\![\cdot]\!]$ on **BExp**: $\alpha \cdot \pi_\alpha \in [\![b]\!] \iff \alpha \leq b \iff \alpha \leq c \iff \alpha \cdot \pi_\alpha \in [\![c]\!]$.

We are now in the position to state and prove soundness of the GNetKAT axioms (which do not include a fixpoint axiom) with respect to the language model.

**Theorem 3.3 (Soundness).** For GNetKAT terms $e$ and $g$, if $e \equiv g$ then $[\![e]\!] = [\![g]\!]$.

*Proof.* By induction on the length of the derivation $e \equiv g$. The full proof is located in the appendix.

## 3.3 Partial Completeness

We now would like to address completeness of the GNetKAT axiomatization according to the language model. Unfortunately, proving full completeness of the GKAT axioms is not an easy matter. The original proof of completeness in [11] assumes an extra *uniqueness axiom* asserting the uniqueness of solutions to certain systems of left-affine equations, in essence a generalized fixpoint rule (W3.). Additionally, the constructions in the original GKAT paper are quite technical and employ coalgebraic theory beyond the scope of this writing. Instead, we aim to give an elementary proof of *loop-free* completeness, which is complete-

ness of the GNetKAT axiomatization with respect to the fragment which has no loops. While not the full axiomatization, a foundational understanding of the language minus loops could help to tackle the problem for the full grammar. Before proceeding, one notion will prove indispensable throughout this section, that of an *atom-indexed* sum. The presentation here is Definition 3.5 in [11].

*Definition 3.4.* For $\Phi \subseteq \mathrm{At}$, let $\{e_\alpha\}_{\alpha\in\Phi}$ be a set of expressions indexed by $\Phi$. The (guarded) sum indexed by $\Phi$ is

$$+_{\alpha\in\Phi} e_\alpha = \begin{cases} e_\beta +_\beta \left(+_{\alpha\in\Phi\setminus\{\beta\}} e_\alpha\right) & \beta \in \Phi \\ 0 & \Phi = \emptyset \end{cases}$$

A frequently employed abuse of notation will be that of replacing $\Phi \subseteq \mathrm{At}$ by a predicate $\Phi(\alpha)$ satisfying $\alpha \in \Phi \iff \Phi(\alpha)$ holds and using $\Phi(\alpha)$ to indicate that we sum over those $\alpha$ satisfying $\Phi(\alpha)$. For example, to sum over all $\alpha \in \mathrm{At}$ we can just sum over those $\alpha \leq 1$. Although it might seem that $+_{\alpha\in\Phi} e_\alpha$ is not well-defined by ambiguity in choice of $\beta$, by skew-associativity and properties of At the expression is indeed uniquely defined up to $\equiv$. See the appendix for more details.

### 3.3.1 GNetKAT to Reduced GNetKAT

Reduced GNetKAT was introduced in Chapter 2 with the intent of aiding in completeness proofs - we expand upon this notion here. Specifically, the syntax of the reduced language closely mirrors the type of the semantics of the full language, which hints that the reduced language could reflect candidate normal

forms. However, we have not yet demonstrated that every GNetKAT program is provably equal to a reduced program. This is captured in Lemma 3.6. First, we need to prove another two statements captured in Lemma 3.5:

*Lemma 3.5.* For $b, c, d \in \mathbf{BExp}$, it holds that $c +_b d \equiv bc \vee \overline{b}d$. Then $+_{\alpha \in \mathrm{At}} \pi_\alpha \equiv 1$ and $+_{\alpha \leq b} 1 \equiv b$ both follow as corollaries.

*Proof.* In order to prove the first statement, first note that $bc \equiv_{\mathbf{BExp}} b(bc \vee \overline{b}d)$ and $\overline{b}d \equiv_{\mathbf{BExp}} \overline{b}(bc \vee \overline{b}d)$. Therefore

$$\begin{aligned}
c +_b d &\equiv bc +_b \overline{b}d && \text{(U4.) (U4′.)} \\
&\equiv b(bc \vee \overline{b}d) +_b \overline{b}(bc \vee \overline{b}d) && \text{(from above)} \\
&\equiv (bc \vee \overline{b}d) +_b (bc \vee \overline{b}d) && \text{(U4.) (U4′.)} \\
&\equiv bc \vee \overline{b}d && \text{(U1.)}
\end{aligned}$$

For the second part of the claim, we proceed by induction on the size of At to show $+_{\alpha \leq 1} 1 \equiv 1$ (this same proof will be adapted to show $b \equiv +_{\alpha \leq b} 1$ for each $b \in \mathbf{BExp}$). Recall that we have $k$ variables, $k \geq 1$ whose values lie in $\{0, 1, \ldots, N\}$ where $N \geq 1$. Therefore the smallest possible size for $|\mathrm{At}|$ is $(1+1)^1 = 2$ according to the general formula $|\mathrm{At}| = (N+1)^k$ with $N = k = 1$. In this case, by the axiom $\vee_{\alpha \in \mathrm{At}} \alpha \equiv 1$ we derive $\alpha_1 \vee \alpha_2 \equiv 1$. Furthermore note that, for $\alpha \neq \beta \in \mathrm{At}$ it holds that $\alpha \cdot (\neg\beta) \equiv \alpha$. This follows since $|\mathrm{At}| \geq 2$, so $\neg\beta \equiv_{\mathbf{BExp}} \vee_{\alpha \neq \beta} \alpha$, from which we left-distribute $\alpha$ and use $\alpha \cdot \gamma \equiv \begin{cases} \alpha & \gamma = \alpha \\ 0 & \gamma \neq \alpha. \end{cases}$. Hence, it holds that

$$+_{\alpha \leq 1} 1 \equiv 1 +_{\alpha_1} (1 +_{\alpha_2} 0) \equiv 1 +_{\alpha_1} \alpha_2 \equiv \alpha_1 \cdot 1 \vee \neg(\alpha_1) \cdot \alpha_2 \equiv \alpha_1 \vee \alpha_2$$

When $|\mathrm{At}| > 2$, let $\beta \in \mathrm{At}$ and find

$$
\begin{aligned}
+_{\alpha\in\mathrm{At}}\,1 &\equiv 1 +_\beta \left(+_{\alpha\in\mathrm{At}\setminus\{\beta\}}1\right) && \text{(def.)}\\
&\equiv 1 +_\beta 1 && \text{(IH)}\\
&\equiv 1 && \text{(U1.)}
\end{aligned}
$$

With the fact $+_{\alpha\in\mathrm{At}}\,1 \equiv 1$ in hand, find

$$
+_{\alpha\in\mathrm{At}}\,1 \equiv +_{\alpha\in\mathrm{At}}\,\alpha \equiv +_{\alpha\in\mathrm{At}}\,\alpha\cdot\pi_\alpha \equiv +_{\alpha\in\mathrm{At}}\,\pi_\alpha
$$

and the lemma is proved. For the third claim $b \equiv +_{\alpha\leq b}1$, we again proceed by induction on size of $\{\alpha \in \mathrm{At} : \alpha \leq b\}$. If this set is empty or a singleton $\{\alpha\}$ then we have $0 \equiv 0$ and $b = \alpha, \alpha \equiv 1 +_\alpha 0$ respectively. The inductive step is the same as above. Specifically, write $b$ as a sum of atoms $b \equiv \sum_{i=1}^{k}\alpha_i$ and apply the inductive step to, say, $\sum_{i=2}^{k}\alpha_i$. □

*Lemma 3.6.* For every $e \in \mathbf{GExp}$ there exists $e' \in \mathbf{GExp}^-$ with $e \equiv e'$.

*Proof.* To demonstrate the lemma, we need to show that every atomic asssignment $f \leftarrow n$ and atomic test $f = n$ can be replaced by provably equivalent reduced terms. First, note

$$
f \leftarrow n \equiv 1\cdot f \leftarrow n \equiv (+_{\alpha\in\mathrm{At}}\,\pi_\alpha)\cdot f \leftarrow n \equiv +_{\alpha\in\mathrm{At}}\,\pi_\alpha[f/n]
$$

which is reduced. For $f = n$, it follows from Lemma 3.6 that $f = n \equiv \vee_{\alpha\leq f=n}\,\alpha$, so $f = n \equiv +_{\alpha\leq f=n}\,1$ by the reasoning as in the previous lemma.

From this, we can express every $b \in \mathbf{BExp}$ as the disjunction of its atoms, substitute every occurrence of $b$ in an expression by its disjunction of atoms,

then use skew-associativity and properties of atoms to convert any occurrence of $e +_b g$ into a guarded sum of atoms (use idempotence on leftmost term and right-associate with skew-associativity). If $b$ appears as a command-level assert then we substitute $b \equiv +_{\alpha \leq b} 1$. For an arbitrary occurrence of $p \triangleq f_{i_1} \leftarrow n_{i_1} \cdot \ldots \cdot f_{i_m} \leftarrow n_{i_m}$ for $1 \leq m \leq k$ and distinct $1 \leq i_1, \ldots, i_m \leq k$ (which we can assume by applying the relevant packet algebra axioms), we imitate the proof for $f \leftarrow n$ shown above. Specifically, write $p \equiv 1 \cdot p \equiv (+_{\alpha \in \mathrm{At}} \pi_\alpha) \cdot p$, right-distribute and use the packet algebra axioms. □

### 3.3.2 Normal Forms

We would now like to devise a normal form for loop-free terms in order to prove loop-free completeness. Before proceeding, we formally define the loop-free terms. The set of loop-free terms $\mathbf{LF} \subseteq \mathbf{GExp}$ is the smallest set satisfying:

(i) each $b \in \mathbf{BExp}$ is in $\mathbf{LF}$;

(ii) primitive actions $f \leftarrow n$ and $\mathbf{dup}$ are in $\mathbf{LF}$;

(iii) if $e, g$ are in $\mathbf{LF}$ then $e +_b g$ and $e \cdot g$ are in $\mathbf{LF}$.

From this definition, complete assignments and tests are also loop-free. We can now describe the normal form for loop-free terms and prove that every loop-free term has a provably equivalent normal form.

*Definition 3.7.* A term $e \in \mathbf{GExp}$ is said to be in *normal form* if it is of the shape $+_{\alpha \in \mathrm{At}} e_\alpha$ where $e_\alpha \in \Pi \cdot (\mathbf{dup} \cdot \Pi)^* \cup \{0\}$ for each $\alpha \in \mathrm{At}$.

**Theorem 3.8.** For every $e \in \mathbf{LF}$ there exists $\{e_\alpha\}_{\alpha \in \mathrm{At}} \subseteq \Pi \cdot (\mathbf{dup} \cdot \Pi)^* \cup \{0\}$ satisfying $e \equiv +_{\alpha \in \mathrm{At}} e_\alpha$. That is, every loop-free term is provably equivalent to a normal form.

*Proof.* By induction on $e$ according to membership in **LF**. The case where $e$ is $f \leftarrow n$ was proven in Lemma 3.6. The case where $e = b \in \mathbf{BExp}$ is also solved easily, since

$$b \equiv_{\mathbf{BExp}} \vee_{\alpha \leq b} \alpha \equiv +_{\alpha \leq b} 1 \equiv +_{\alpha \leq b} \pi_\alpha \equiv +_{\alpha \in \mathrm{At}} m_\alpha$$

where $m_\alpha \triangleq \begin{cases} \pi_\alpha & \alpha \leq b \\ 0 & \alpha \not\leq b \end{cases} \in \Pi \cdot (\mathbf{dup} \cdot \Pi)^* \cup \{0\}$. For the case where $e$ is **dup**,

$$\begin{aligned}
\mathbf{dup} &\equiv 1 \cdot \mathbf{dup} \\
&\equiv (+_{\alpha \in \mathrm{At}} \pi_\alpha) \cdot \mathbf{dup} \\
&\equiv +_{\alpha \in \mathrm{At}} (\pi_\alpha \cdot \mathbf{dup}) \\
&\equiv +_{\alpha \in \mathrm{At}} (\pi_\alpha \cdot \alpha \cdot \mathbf{dup}) \\
&\equiv +_{\alpha \in \mathrm{At}} (\pi_\alpha \cdot \mathbf{dup} \cdot \alpha) \\
&\equiv +_{\alpha \in \mathrm{At}} (\pi_\alpha \cdot \mathbf{dup} \cdot \alpha \cdot \pi_\alpha) \\
&\equiv +_{\alpha \in \mathrm{At}} (\pi_\alpha \cdot \alpha \cdot \mathbf{dup} \cdot \pi_\alpha) \\
&\equiv +_{\alpha \in \mathrm{At}} (\pi_\alpha \cdot \mathbf{dup} \cdot \pi_\alpha)
\end{aligned}$$

The two cases which remain are that of $e = g +_b h$ and $e = g \cdot h$ for $g, h \in \mathbf{LF}$. We need to show that, in each case (assuming inductive hypotheses) $e$ is provably equivalent to an expression of shape $+_{\alpha \in \mathrm{At}} e_\alpha$ where $e_\alpha \in \Pi \cdot (\mathbf{dup} \cdot \Pi)^* \cup \{0\}$ for each $\alpha \in \mathrm{At}$.

Case: $e = g +_b h$. Let $g \equiv +_{\alpha\in\mathrm{At}}\, g_\alpha$ and $h \equiv +_{\alpha\in\mathrm{At}}\, h_\alpha$ according to the IH. Set $E = \{\alpha \in \mathrm{At} : \alpha \leq b\}$. It follows that $b \equiv_{\mathbf{BExp}} \vee_{\alpha\in E}\, \alpha \equiv +_{\alpha\in E}\, 1$ and $\overline{b} \equiv +_{\alpha\in\overline{E}}\, 1$, so $b\cdot k \equiv +_{\alpha\in E}\, k$ for any $k \in \mathbf{GExp}$ by applying right-distributivity (U5.) and neutral left (S4.) and likewise for $\overline{b}$. Therefore

$$\begin{aligned}
g +_b h &\equiv (+_{\alpha\in\mathrm{At}}\, g_\alpha) +_b (+_{\alpha\in\mathrm{At}}\, h_\alpha) \\
&\equiv (b \cdot +_{\alpha\in\mathrm{At}}\, g_\alpha) +_b \left(\overline{b} \cdot +_{\alpha\in\mathrm{At}}\, h_\alpha\right) \\
&\equiv \left(+_{\beta\in E}\, +_{\alpha\in\mathrm{At}}\, g_\alpha\right) +_b \left(+_{\beta\in\overline{E}}\, +_{\alpha\in\mathrm{At}}\, h_\alpha\right) \\
&\equiv \left(+_{\beta\in E}\, \beta \cdot +_{\alpha\in\mathrm{At}} g_\alpha\right) +_b \left(+_{\beta\in\overline{E}} \beta \cdot +_{\alpha\in\mathrm{At}}\, h_\alpha\right) \\
&\equiv \left(+_{\beta\in E}\, \beta \cdot g_\beta\right) +_b \left(+_{\beta\in\overline{E}} \beta \cdot h_\beta\right) \\
&\equiv \left(+_{\beta\in E}\, g_\beta\right) +_b \left(+_{\beta\in\overline{E}} h_\beta\right)
\end{aligned}$$

by applying guardedness (U4.) and branch choice (U8.). Now, for each $\alpha \in \mathrm{At}$ set $e_\alpha \triangleq \begin{cases} g_\alpha & \alpha \in E \\ h_\alpha & \alpha \notin E \end{cases}$. Then, it follows that

$$\begin{aligned}
+_{\alpha\in\mathrm{At}}\, e_\alpha &\equiv (+_{\alpha\in\mathrm{At}}\, e_\alpha) +_b (+_{\alpha\in\mathrm{At}}\, e_\alpha) \\
&\equiv (b \cdot +_{\alpha\in\mathrm{At}}\, e_\alpha) +_b \left(\overline{b} \cdot +_{\alpha\in\mathrm{At}}\, e_\alpha\right) \\
&\equiv \left(+_{\beta\in E}\, \beta \cdot +_{\alpha\in\mathrm{At}}\, e_\alpha\right) +_b \left(+_{\beta\in\overline{E}} \beta \cdot +_{\alpha\in\mathrm{At}}\, e_\alpha\right) \\
&\equiv \left(+_{\beta\in E}\, e_\beta\right) +_b \left(+_{\beta\in\overline{E}}\, e_\beta\right) \\
&= \left(+_{\beta\in E}\, g_\beta\right) +_b \left(+_{\beta\in\overline{E}}\, h_\beta\right)
\end{aligned}$$

where the final equality follows by definition of the $e_\alpha$'s. Since we already showed $g +_b h \equiv \left(+_{\beta\in E}\, g_\beta\right) +_b \left(+_{\beta\in\overline{E}}\, h_\beta\right)$, we are done in this case.

Case: $e = g \cdot h$. Let $g \equiv +_{\alpha\in\mathrm{At}}\, g_\alpha$ and $h \equiv +_{\alpha\in\mathrm{At}}\, h_\alpha$ by the IH. For each $\alpha \in \mathrm{At}$ let $\pi_\gamma$ be the trailing complete assignment in $g_\alpha$, where $\pi_\gamma$ corresponds with $\gamma = \gamma(\alpha) \in \mathrm{At}$ (recall the IH says $g_\alpha \in \Pi \cdot (\mathbf{dup} \cdot \Pi)^* \cup \{0\} = (\Pi \cdot \mathbf{dup})^* \cdot \Pi \cup \{0\}$) and write $g_\alpha = x_\alpha \cdot \pi_\gamma$ for some $x_\alpha \in (\Pi \cdot \mathbf{dup})^*$. We have

$$
\begin{aligned}
g \cdot h &\equiv (+_{\alpha \in \mathrm{At}}\, g_\alpha) \cdot (+_{\alpha \in \mathrm{At}}\, h_\alpha) \\
&\equiv +_{\alpha \in \mathrm{At}} \left( g_\alpha \cdot +_{\beta \in \mathrm{At}}\, h_\beta \right) \\
&\equiv +_{\alpha \in \mathrm{At}} \left( x_\alpha \cdot \pi_\gamma \cdot +_{\beta \in \mathrm{At}}\, h_\beta \right) \\
&\equiv +_{\alpha \in \mathrm{At}} \left( x_\alpha \cdot \pi_\gamma \cdot \gamma \cdot +_{\beta \in \mathrm{At}}\, h_\beta \right) \\
&\equiv +_{\alpha \in \mathrm{At}} \left( x_\alpha \cdot \pi_\gamma \cdot \gamma \cdot h_\gamma \right) \\
&\equiv +_{\alpha \in \mathrm{At}} \left( x_\alpha \cdot \pi_\gamma \cdot h_\gamma \right) \\
&\equiv +_{\alpha \in \mathrm{At}} \left( g_\alpha \cdot h_\gamma \right)
\end{aligned}
$$

We are almost done, except that the concatenations $g_\alpha \cdot h_\gamma$ (recall $\gamma$ depends on $\alpha$) are not quite of the right shape. Therefore define a modified concatenation operator $\otimes : (\Pi \cdot (\mathbf{dup} \cdot \Pi)^* \cup \{0\})^2 \to \Pi \cdot (\mathbf{dup} \cdot \Pi)^* \cup \{0\}$ as $x \otimes y \triangleq \begin{cases} p \cdot \pi_\eta \cdot q & \text{if } x = p \cdot \pi_\gamma, y = \pi_\eta \cdot q \\ 0 & \text{otherwise} \end{cases}$ where we again use the equality $\Pi \cdot (\mathbf{dup} \cdot \Pi)^* = (\Pi \cdot \mathbf{dup})^* \cdot \Pi$ (note $p \in (\Pi \cdot \mathbf{dup})^*$ and $q \in (\mathbf{dup} \cdot \Pi)^*$). It is a straightforward proof by case analysis (no induction necessary) that $x \otimes y \equiv x \cdot y$. The only non-trivial case is $p \cdot \pi_\gamma \otimes \pi_\eta \cdot q \triangleq p \cdot \pi_\eta \cdot q \equiv p \cdot (\pi_\gamma \cdot \pi_\eta) \cdot q \equiv (p \cdot \pi_\gamma) \cdot (\pi_\eta \cdot q)$. Therefore, since we know that both $g_\alpha, h_\gamma \in \Pi \cdot (\mathbf{dup} \cdot \Pi)^* \cup \{0\}$, it follows that $g_\alpha \otimes h_\gamma \in \Pi \cdot (\mathbf{dup} \cdot \Pi)^* \cup \{0\}$ and $g_\alpha \otimes h_\gamma \equiv g_\alpha \cdot h_\gamma$. This lets us proceed as

$$
g \cdot h \equiv +_{\alpha \in \mathrm{At}} (g_\alpha \cdot h_\gamma) \equiv +_{\alpha \in \mathrm{At}} (g_\alpha \otimes h_\gamma)
$$

We are done in all cases. □

### 3.3.3 Loop-free Completeness

We prove the final ingredient for loop-free completeness, that being that the axioms are complete for normal forms.

**Theorem 3.9 (Normal Form Completeness).** For two loop-free normal forms $+_{\alpha\in\mathrm{At}}\, e_\alpha$ and $+_{\alpha\in\mathrm{At}}\, g_\alpha$, if $[\![+_{\alpha\in\mathrm{At}}\, e_\alpha]\!] = [\![+_{\alpha\in\mathrm{At}}\, g_\alpha]\!]$ then $+_{\alpha\in\mathrm{At}}\, e_\alpha \equiv +_{\alpha\in\mathrm{At}}\, g_\alpha$.

*Proof.* With the assumption $[\![+_{\alpha\in\mathrm{At}}\, e_\alpha]\!] = [\![+_{\alpha\in\mathrm{At}}\, g_\alpha]\!]$, we will prove in fact that $e_\beta = g_\beta$ for each $\beta \in \mathrm{At}$, from which the result follows (recall that atom-guarded sums are only defined up to $\equiv$). Let $\beta \in \mathrm{At}$. We have two cases to consider.

If there exists $x \in (\Pi \cdot \mathbf{dup})^*$ and $\pi \in \Pi$ for which $\beta \cdot x \cdot \pi \in [\![+_{\alpha\in\mathrm{At}}\, e_\alpha]\!] = [\![+_{\alpha\in\mathrm{At}}\, g_\alpha]\!]$, then necessarily $\beta \cdot x \cdot \pi \in [\![e_\beta]\!]$ and $\beta \cdot x \cdot \pi \in [\![g_\beta]\!]$. Clearly $e_\beta, g_\beta \neq 0$ under these assumptions, so $e_\beta, g_\beta \in (\Pi \cdot \mathbf{dup})^* \cdot \Pi$. Also, it is easily proven by induction on $z \in (\Pi \cdot \mathbf{dup})^* \cdot \Pi$ that $[\![z]\!] = \{\alpha \cdot z : \alpha \in \mathrm{At}\}$ for all $z$. Therefore, $\beta \cdot x \cdot \pi \in [\![e_\beta]\!]$ if and only if $x \cdot \pi = e_\beta$ and likewise for $g_\beta$. Therefore $e_\beta = x \cdot \pi = g_\beta$, so the claim holds in this case.

If there is no $x \in (\Pi \cdot \mathbf{dup})^*$, $\pi \in \Pi$ for which $\beta \cdot x \cdot \pi \in [\![+_{\alpha\in\mathrm{At}}\, e_\alpha]\!] = [\![+_{\alpha\in\mathrm{At}}\, g_\alpha]\!]$, then necessarily there is no $x \in (\Pi \cdot \mathbf{dup})^*$, $\pi \in \Pi$ for which either $\beta \cdot x \cdot \pi \in [\![e_\beta]\!]$ or $\beta \cdot x \cdot \pi \in [\![g_\beta]\!]$ holds. This holds if and only if $e_\beta = 0 = g_\beta$, since otherwise $\beta \cdot e_\beta \in [\![e_\beta]\!]$ and likewise for $g_\beta$. Therefore the claim holds in this case.

Since the claim holds in all cases for each $\beta \in \mathrm{At}$, we have $e_\beta = g_\beta$ for each $\beta \in \mathrm{At}$ and thus $+_{\alpha\in\mathrm{At}}\, e_\alpha \equiv +_{\alpha\in\mathrm{At}}\, g_\alpha$. □

We can now prove the following theorem.

**Theorem 3.10 (Loop-free Completeness).** Let $e, g \in \mathbf{LF}$. Then $[\![e]\!] = [\![g]\!]$ implies

$e \equiv g$.

*Proof.* Let $e \equiv +_{\alpha \in \mathrm{At}} e_\alpha$ and $g \equiv +_{\alpha \in \mathrm{At}} g_\alpha$ by Theorem 3.8. By soundness, $[\![e]\!] = [\![+_{\alpha \in \mathrm{At}} e_\alpha]\!]$ and $[\![g]\!] = [\![+_{\alpha \in \mathrm{At}} g_\alpha]\!]$, so by our assumptions $[\![+_{\alpha \in \mathrm{At}} e_\alpha]\!] = [\![+_{\alpha \in \mathrm{At}} g_\alpha]\!]$. By Theorem 3.9, it follows that $+_{\alpha \in \mathrm{At}} e_\alpha \equiv +_{\alpha \in \mathrm{At}} g_\alpha$, from which

$$e \equiv +_{\alpha \in \mathrm{At}} e_\alpha \equiv +_{\alpha \in \mathrm{At}} g_\alpha \equiv g$$

as claimed. □

We have thus accomplished what we set out to do: proving soundness and completeness over the fragment of GNetKAT which does not mention loops. As discussed in Section 3.1, full soundness and completeness for GNetKAT is a hopeless endeavor until one develops a better understanding of the empty word property and the correct axiomatization for loops in this setting.

CHAPTER 4

# COALGEBRAIC DECISION PROCEDURE

After studying the relationship between the semantics and axioms, we now would like a general method to decide whether two GNetKAT programs $e$ and $g$ are equivalent or not. Specifically, we aim to provide a nearly linear algorithm (in the size of $e$ and $g$) which can decide language equivalence of $e$ and $g$. This goal is taken from the previous accomplishments in [11], where uninterpreted GKATs semantic equivalence procedure was shown to be $O(n \cdot \alpha(n))$ where $\alpha(n) = A^{-1}(n, n)$ is the inverse Ackermann function. The route we take is similar to theirs. The type of a GNetKAT coalgebra is specified via a functor, then we define a subclass of coalgebras where bisimilarity coincides with language equivalence, then describe the final coalgebra in the relevant category of objects. From this, we can give a Hopcroft-Karp style algorithm which makes use of the efficient union-find data structure.

## 4.1 $H$-Coalgebras

Define a functor $H : \mathbf{Set} \to \mathbf{Set}$ by $HX = (1 + \mathrm{At} + \mathrm{At} \times X)^{\mathrm{At}}$, where $1 \triangleq \{\bot\}$ is any singleton set and $+$ is coproduct (disjoint union). An $H$-coalgebra is a pair $\mathcal{X} = (X, \delta_{\mathcal{X}} : X \to H(X))$, with $X$ the *state space* and *transition map* $\delta_{\mathcal{X}} : X \to H(X)$. Given a state $x \in X$ and an $\alpha \in \mathrm{At}$, one of three possibilities may occur in the coalgebra: it either crashes (halts and rejects) on its input, halts successfully and outputs $\beta \in \mathrm{At}$, or yields a new state $x$ and a new atom $\beta$. We suppress canonical injections and write either $\delta_{\mathcal{X}}(x)(\alpha) = \bot$ or $\delta_{\mathcal{X}}(x)(\alpha) = \beta$ or $\delta_{\mathcal{X}}(x)(\alpha) = (\beta, x)$ according to the three mutually exclusive possibilities.

We can interpret $H$-coalgebras as acceptors of guarded traces. Acceptance for an $H$-coalgebra $(X, \delta_X)$ is defined for a state $x \in X$ by

$$\begin{aligned} \text{accept}(x, \alpha \cdot \pi_\beta) &\iff \delta_X(x)(\alpha) = \beta \\ \text{accept}(x, \alpha \cdot \pi_\beta \cdot \mathbf{dup} \cdot w) &\iff \exists y.\, \delta_X(x)(\alpha) = (\beta, y) \wedge \text{accept}(y, \beta \cdot w) \end{aligned}$$

Notice that the acceptance condition enforces that the output atom of one state is the input atom for the next. The language of (finite) guarded strings accepted by a state $x \in X$ is the set of all $w \in \text{GT}$ for which $\text{accept}(x, w)$, denoted by $l^X(x) \subseteq \text{GT}$. An *$H$-automaton* $(X, \delta_X, x_0)$ is an $H$-coalgebra $(X, \delta_X)$ equipped with an *initial state* $x_0 \in X$. We are interested in casting expressions $e \in \mathbf{GExp}$ as $H$-automata to leverage a fast equivalence procedure by testing bisimilarity of start states. To do so, we first equip both the set of deterministic languages $\mathcal{L} \subseteq 2^{\text{GT}}$ and the set $\mathbf{GExp}$ with the structure of $H$-coalgebras. These will be known, respectively, as the *semantic* coalgebra and the *syntactic* coalgebra.

The semantic $H$-coalgebra is defined over the set of deterministic languages $\mathcal{L} \subseteq 2^{\text{GT}}$, where $\mathcal{L}$ is the collection of all languages $L \subseteq \text{GT}$ for which each $\alpha \in \text{At}$ is the prefix of at most one string in $L$. The semantic $H$-coalgebra is given by $(\mathcal{L}, \delta_{\mathcal{L}})$ where $\delta_{\mathcal{L}} : \mathcal{L} \to H(\mathcal{L})$ is defined as as follows:

$$\delta_{\mathcal{L}}(L)(\alpha) = \begin{cases} \bot & \text{if } \{y \in \Pi \cdot (\mathbf{dup} \cdot \Pi)^* : \alpha \cdot y \in L\} = \emptyset \\ \beta & \text{if } \alpha \cdot \pi_\beta \in L \\ (\beta, \{\beta \cdot x\}) & \text{if } \alpha \cdot \pi_\beta \cdot \mathbf{dup} \cdot x \in L \end{cases}$$

By the determinacy property of languages $L \in \mathcal{L}$, the map $\delta_{\mathcal{L}} : \mathcal{L} \to H(\mathcal{L})$ is well-defined: every $\alpha \in \text{At}$ is the prefix of at most one string in $L$. If $\alpha$ is the

prefix of a unique string $x$ belonging to $L$ (by determinacy of $L$), then the string $x$ is either of shape $\alpha \cdot \pi$ or $\alpha \cdot \pi \cdot \mathbf{dup} \cdot y$ for $y \in \Pi \cdot (\mathbf{dup} \cdot \Pi)^*$ and not both. This exhausts the possibilities for every $\alpha \in \mathrm{At}$, so the map $L \mapsto \delta_{\mathcal{L}}(L)$ is well-defined. Furthermore, notice that in the semantic coalgebra the only transitions of shape $\delta_{\mathcal{L}}(L)(\alpha) = (\beta, L')$ satisfy that $L'$ is a singleton set. This is a consequence of the determinacy property satisfied by GNetKAT languages (Definition 2.1), but it may seem unnatural at first glance, especially since the analogous semantic coalgebra in abstract GKAT is not so simple [11]. This property of the semantic coalgebra will be known later as *strong normality*.

We now turn to define the syntactic $H$-coalgebra. We define the map $D : \mathbf{GExp} \to H(\mathbf{GExp})$ by structural recursion on $e \in \mathbf{GExp}$ in Figure 4.1. We note that the only time transitions are taken between states is when we encounter a **dup**. Furthermore the definition for $D(e^{(b)})$ is circular, but one makes sense of it by taking least fixed points in the directed-complete partial order $(\{\delta : \mathbf{GExp} \to H(\mathbf{GExp})\}, \prec)$ where $\delta_1 \prec \delta_2$ if and only if for all $e \in \mathbf{GExp}$, $\delta_2(e)$ extends $\delta_1(e)$ as partial functions $\mathrm{At} \rightharpoonup \mathrm{At} + \mathrm{At} \times \mathbf{GExp}$, where we interpret $\delta(e)(\alpha) = \bot$ as $\delta(e)(\alpha)$ as not being defined.

We will soon relate the $H$-coalgebras $(\mathbf{GExp}, D)$ and $(\mathcal{L}, \delta)$, in particular that two states in $(\mathbf{GExp}, D)$ are bisimilar if and only if they are language equivalent. Unfortunately only the forward direction is true for general $H$-coalgebras, due to the possibility for a coalgebra to reject late rather than as soon as possible. To address this, we take a detour to describe *normal* $H$-coalgebras similar to (but different from) the presentation in [11], followed by the novel concept of strong normality satisfied by GNetKAT expressions.

$$
\begin{aligned}
D(f \leftarrow n)(\alpha) &\triangleq \alpha' \text{ where } \alpha' = \alpha[f/n] \\
D(b)(\alpha) &\triangleq \begin{cases} \alpha & \alpha \leq b \\ \bot & \alpha \not\leq b \end{cases} \\
D(\mathbf{dup})(\alpha) &\triangleq (\alpha, \alpha) \\
D(e +_b g)(\alpha) &\triangleq \begin{cases} D(e)(\alpha) & \alpha \leq b \\ D(g)(\alpha) & \alpha \not\leq b \end{cases} \\
D(e \cdot g)(\alpha) &\triangleq \begin{cases} \bot & D(e)(\alpha) = \bot \\ D(g)(\beta) & D(e)(\alpha) = \beta \\ (\beta, e' \cdot g) & D(e)(\alpha) = (\beta, e') \end{cases} \\
D(e^{(b)})(\alpha) &\triangleq \begin{cases} \alpha & \alpha \not\leq b \\ (\beta, e' \cdot e^{(b)}) & \alpha \leq b \wedge D(e)(\alpha) = (\beta, e') \\ \bot & \alpha \leq b \wedge D(e)(\alpha) = \bot \\ D(e^{(b)})(\beta) & \alpha \leq b \wedge D(e)(\alpha) = \beta \end{cases}
\end{aligned}
$$

Figure 4.1: Syntactic derivative for **GExp**

## 4.2 (Strongly) Normal $H$-coalgebras

We begin with the following definition, which contains modified versions of the liveness properties described in [11].

*Definition 4.1 (Live, Dead, Normal).* Let $\mathcal{X} = (X, \delta_X)$ be an $H$-coalgebra. A state $x \in X$ is *live* if there is some $w \in \mathrm{GT}$ for which $w \in l^{\mathcal{X}}(x)$, otherwise it is *dead*. An $H$-coalgebra is *normal* if it has no transitions into dead states (although it may contain dead states).

Note that an equivalent condition for $x \in X$ being live is that $l^{\mathcal{X}}(x) \neq \emptyset$, and an equivalent condition for $x$ being dead is that $l^{\mathcal{X}}(x) = \emptyset$. We can normalize every $H$-coalgebra $\mathcal{X} = (X, \delta_{\mathcal{X}})$ to transform it into a normal $H$-coalgebra $\tilde{\mathcal{X}} = (X, \delta_{\tilde{\mathcal{X}}})$ over the same state space. To do so, let $D \subseteq X$ denote the set of dead states and set

$$\delta_{\tilde{\mathcal{X}}}(x)(\alpha) \triangleq \begin{cases} \bot & \delta_{\mathcal{X}}(x)(\alpha) \in \mathrm{At} \times D \\ \delta_{\mathcal{X}}(x)(\alpha) & \text{otherwise} \end{cases}$$

*Lemma 4.2 (Correctness of normalization).* Let $\mathcal{X} = (X, \delta_X)$ be an $H$-coalgebra. Then

(i) $\mathcal{X}$ and $\tilde{X}$ accept the same languages, that is, $l^{\mathcal{X}} = l^{\tilde{\mathcal{X}}}$;

(ii) $\tilde{\mathcal{X}}$ is normal.

*Proof.* For (i), we show for all $x \in X$ that $l^{\mathcal{X}}(x) = l^{\tilde{\mathcal{X}}}(x)$ holds by proving $t \in l^{\mathcal{X}}(x) \iff t \in l^{\tilde{\mathcal{X}}}(x)$ by induction on the size of $t \in \mathrm{GT} = \mathrm{At} \cdot \Pi \cdot (\mathbf{dup} \cdot \Pi)^*$. Let accept denote the acceptance predicate for $\mathcal{X}$ and accept′ for $\tilde{\mathcal{X}}$. Let $t = \alpha \cdot \pi_\beta \in l^{\mathcal{X}}(x)$. Then $\delta_{\mathcal{X}}(x)(\alpha) = \beta = \delta_{\tilde{\mathcal{X}}}(x)(\alpha)$ by definition so $t \in \delta_{\tilde{\mathcal{X}}}(x)(\alpha)$. Conversely let $t \in l^{\tilde{\mathcal{X}}}(x)$. Then $\delta_{\tilde{\mathcal{X}}}(x)(\alpha) = \beta = \delta_{\mathcal{X}}(x)(\alpha)$ by definition so $t \in l^{\mathcal{X}}(x)$.

When $t = \alpha \cdot \pi_\beta \cdot \mathbf{dup} \cdot p$ for $p \in \Pi \cdot (\mathbf{dup} \cdot \Pi)^*$, consider

$$\begin{aligned}
\mathrm{accept}(x, \alpha \cdot \pi_\beta \cdot \mathbf{dup} \cdot p) &\iff \exists y.\, \delta_{\mathcal{X}}(x)(\alpha) = (\beta, y) \wedge \mathrm{accept}(y, \beta \cdot p) && \text{(def.)} \\
&\iff \exists y \notin D.\, \delta_{\mathcal{X}}(x)(\alpha) = (\beta, y) \wedge \mathrm{accept}(y, \beta \cdot p) && (*) \\
&\iff \exists y \notin D.\, \delta_{\tilde{\mathcal{X}}}(x)(\alpha) = (\beta, y) \wedge \mathrm{accept}'(y, \beta \cdot p) && \text{(IH)} \\
&\iff \mathrm{accept}'(x, \alpha \cdot \pi_\beta \cdot \mathbf{dup} \cdot p) && \text{(def.)}
\end{aligned}$$

where $(*)$ uses the definition of $D$ the set of dead states in $X$: for $y$ to satisfy $\mathrm{accept}(y, \beta \cdot p)$ it must be that $y \notin D$, so $\delta_{\mathcal{X}}(x)(\alpha) = (\beta, y) = \delta_{\tilde{\mathcal{X}}}(x)(\alpha)$. Therefore claim (i) is proven.

To see claim (ii), note that $x \in X$ is dead in $\mathcal{X}$ if and only if it is dead in $\tilde{\mathcal{X}}$ by claim (i), since $x$ is dead if and only if $l^{\mathcal{X}}(x) = l^{\tilde{\mathcal{X}}}(x) = \emptyset$. Therefore, by construction, $\tilde{\mathcal{X}}$ has no transitions into dead states, so $\tilde{\mathcal{X}}$ is normal. □

Notice that the semantic coalgebra as defined above is already normal: a state $L \in \mathcal{L}$ is dead if and only if it is empty, and there are no transitions to the empty language by construction. To see this first claim, we first show that $L \neq \emptyset$ implies $L$ is live by proving for all $w$ that $w \in L$ implies $L$ is live, by induction on $w$. If $w = \alpha \cdot \pi_\beta \in L$ then $\delta_{\mathcal{L}}(L)(\alpha) = \beta$ so $w \in l^{\mathcal{L}}(L)$; if $w = \alpha \cdot \pi_\beta \cdot \mathbf{dup} \cdot w' \in L$ then $\delta_{\mathcal{L}}(L)(\alpha) = (\beta, \{\beta \cdot w'\})$, so by induction $\{\beta \cdot w'\}$ must accept some string; the only candidate is $\beta \cdot w'$, from which $\alpha \cdot \pi_\beta \cdot \mathbf{dup} \cdot w' \in l^{\mathcal{L}}(L)$ follows. The converse, that $L$ live implies $L \neq \emptyset$, is argued analogously by induction on the witness $w \in l^{\mathcal{L}}(L)$. Therefore $L \in \mathcal{L}$ is dead if and only if $L = \emptyset$. As it turns out, the operational properties of the syntactic and semantic coalgebra for GNetKAT are actually a bit stronger. This is captured in the next definition.

*Definition 4.3.* An $H$-coalgebra $(X, \delta_X)$ is said to be *strongly normal* if whenever $\delta_X(s)(\alpha) = (\beta, t)$ holds for some $s, t \in X$, $\alpha, \beta \in \mathrm{At}$ then $l^{\mathcal{X}}(t) = \{\beta \cdot w\}$ for some $w \in \Pi \cdot (\mathbf{dup} \cdot \Pi)^*$.

It is immediate from the definition that any strongly normal $H$-coalgebra is also normal, hence the name. The semantic coalgebra $(\mathcal{L}, \delta_{\mathcal{L}})$ is strongly normal, which gives justification to its unusual shape. It is also easy to show that the normalized syntactic coalgebra $(\mathbf{GExp}, \tilde{D})$ is strongly normal. As a brief sketch, in the normal coalgebra $(\mathbf{GExp}, \tilde{D})$ one proves by induction that if $\tilde{D}(e)(\alpha) = (\beta, e')$ holds then either $e' = \beta \cdot e''$ for some $e''$ or $e' = \beta$ (where $e'$ is live), i.e. we only ever step because we executed a $\mathbf{dup}$, with the base case of the induction being $\tilde{D}(\mathbf{dup})(\gamma) = (\gamma, \gamma)$ for any $\gamma \in \mathrm{At}$. Therefore any string $w \in \mathrm{GT}$ which

satisfies accept$(e', w)$ (one must exist by liveness) must begin with $\beta$, from which it follows by determinism that $l^D(e') = \{w\} = \{\beta \cdot w'\}$ for some $w' \in \Pi \cdot (\mathbf{dup} \cdot \Pi)^*$, i.e. strong normality of $(\mathbf{GExp}, \tilde{D})$ holds.

Strong normality is an important property as will be explained in the next section. Consider the following $H$-coalgebra $\mathcal{X} = (X, \delta_\mathcal{X})$ which is normal but not strongly normal:

$$X = \{s, r, t, p, q\}$$

$$\delta_\mathcal{X}(s)(\alpha) = (\beta, t) \qquad \delta_\mathcal{X}(r)(\alpha) = (\gamma, t)$$

$$\delta_\mathcal{X}(t)(\beta) = (\delta, p) \qquad \delta_\mathcal{X}(t)(\gamma) = (\delta, q)$$

$$\delta_\mathcal{X}(p)(\delta) = \delta_\mathcal{X}(q)(\delta) = \delta \qquad \text{all other transitions are } \bot$$

This $H$-coalgebra is normal by the mere fact that no state is dead. It is not strongly normal, since $\delta_\mathcal{X}(s)(\alpha) = (\beta, t)$ but $l^\mathcal{X}(t) = \{\beta \cdot \pi_\delta \cdot \mathbf{dup} \cdot \pi_\delta, \gamma \cdot \pi_\delta \cdot \mathbf{dup} \cdot \pi_\delta\}$. In fact, this example demonstrates that if $\mathcal{X} = (X, \delta_\mathcal{X})$ is not strongly normal, then the language acceptance map $l^\mathcal{X} : X \to \mathcal{L}$ may not be a *homomorphism*, whose definition we provide in the next section. Intuitively, even though $\delta_\mathcal{X}(s)(\alpha) = (\beta, t)$ holds in $\mathcal{X}$, it is not true that $\delta_\mathcal{L}(l^\mathcal{X}(s)) = (\beta, l^\mathcal{X}(t))$ in $(\mathcal{L}, \delta_\mathcal{L})$, since

$$l^\mathcal{X}(s) = \{\alpha \cdot \pi_\beta \cdot \mathbf{dup} \cdot \pi_\delta \cdot \mathbf{dup} \cdot \pi_\delta\} \qquad l^\mathcal{X}(t) = \{\beta \cdot \pi_\delta \cdot \mathbf{dup} \cdot \pi_\delta, \gamma \cdot \pi_\delta \cdot \mathbf{dup} \cdot \pi_\delta\}$$

Strong normality is precisely the condition needed to address this issue. One may suggest altering the definition of the semantic coalgebra to do away with the requirement of strong normality, but this is not so easy either due to the shape of the acceptance condition. That is, simply knowing $\delta_\mathcal{X}(s)(\alpha) = (\beta, t)$ and

$l^{\mathcal{X}}(s)$ is not enough to compute $l^{\mathcal{X}}(t)$, since $l^{\mathcal{X}}(t)$ could allow for arbitrary starting atoms, whereas the paths from $s$ through $t$ in the coalgebra only contribute to language acceptance of $s$ if the corresponding output-input atoms match. The assumption of strong normality enforces that the $H$-coalgebras of interest behave like the semantic coalgebra, in particular enforcing that the output-as-next-input scheme completely defines the edges of the labelled transition system arising from the coalgebra. In other words, only distinguished "start states" $s \in X$ (those with no incoming transitions) can ever satisfy $|l^{\mathcal{X}}(s)| > 1$.

## 4.3 Homomorphisms, Bismulations, Algorithm

We formally define the notions of bisimilarity and homomorphism for $H$-coalgebras. These concepts will play a key role in the decision procedure, as we aim to establish bisimilarity between states of an $H$-coalgebra by showing they have the same image under the unique homomorphism into the final object in the category of strongly normal $H$-coalgebras.

*Definition 4.4 (Bisimulation).* Let $\mathcal{X} = (X, \delta_{\mathcal{X}})$ and $\mathcal{Y} = (Y, \delta_{\mathcal{Y}})$. A *bisimulation* between $\mathcal{X}$ and $\mathcal{Y}$ is a relation $R \subseteq X \times Y$ such that, if $xRy$ holds then:

(i) if $\delta_{\mathcal{X}}(x)(\alpha) \in 1 + \mathrm{At}$ then $\delta_{\mathcal{Y}}(y)(\alpha) = \delta_{\mathcal{X}}(x)(\alpha)$;

(ii) if $\delta_{\mathcal{X}}(x)(\alpha) = (\beta, x')$ then there exists some $y' \in Y$ for which $\delta_{\mathcal{Y}}(y)(\alpha) = (\beta, y')$ and $x'Ry'$ holds.

Two states $x \in X, y \in Y$ are bisimilar, written $x \sim y$, if there exists a bisimulation containing $(x, y)$. Despite (i) and (ii) above being implications, it is a standard

result that the bisimilarity relation $\sim$ is an equivalence relation [7]. Furthermore, one easily proves that bisimilar states are language equivalent:

*Lemma 4.5.* If $\mathcal{X}$ and $\mathcal{Y}$ are $H$-coalgebras with bisimilar states $x \sim y$ then $l^{\mathcal{X}}(x) = l^{\mathcal{Y}}(y)$.

The proof of this lemma is found in the appendix. Next, we need to define the notion of a structure-preserving map between $H$-coalgebras.

*Definition 4.6 (Homomorphism).* A *homomorphism* between $H$-coalgebras $\mathcal{X}$ and $\mathcal{Y}$ is a function $h : X \to Y$ which satisfies, for all $x, x' \in X$, $\alpha, \beta \in \mathrm{At}$:

(i) if $\delta_{\mathcal{X}}(x)(\alpha) \in 1 + \mathrm{At}$ then $\delta_{\mathcal{Y}}(h(x))(\alpha) = \delta_{\mathcal{X}}(x)(\alpha)$;

(ii) if $\delta_{\mathcal{X}}(x)(\alpha) = (\beta, x')$ then $\delta_{\mathcal{Y}}(h(x))(\alpha) = (\beta, h(x'))$.

Note that one can prove from this definition that homomorphisms also *reflect* transitions as well: if $\delta_{\mathcal{Y}}(h(x))(\alpha) = (\beta, t)$ then there exists some $x' \in X$ for which $\delta_{\mathcal{X}}(x)(\alpha) = (\beta, x')$ and $h(x') = t$ [7]. We are now in the position to prove an important theorem, that being that the semantic coalgebra $(\mathcal{L}, \delta_{\mathcal{L}})$ is final in the category of strongly normal $H$-coalgebras, whose objects are strongly normal $H$-coalgebras and morphisms are homomorphisms of $H$-coalgebras. The proof of Theorem 4.7 is in the appendix.

**Theorem 4.7.** If $\mathcal{X} = (X, \delta_{\mathcal{X}})$ is a strongly normal $H$-coalgebra, then $l^{\mathcal{X}} : X \to \mathcal{L}$ is the unique homomorphism from $\mathcal{X}$ to $(\mathcal{L}, \delta_{\mathcal{L}})$.

The following corollary is now a standard result in universal coalgebra [7].

*Corollary 4.8.* Let $\mathcal{X}$ and $\mathcal{Y}$ be strongly normal $H$-coalgebras. Then $x \sim y$ if and only if $l^{\mathcal{X}}(x) = l^{\mathcal{Y}}(y)$.

*Proof.* The forward direction is Lemma 4.5. For the reverse direction, define $R \triangleq \{(s,t) \in X \times Y : l^X(s) = l^Y(t)\}$. We will show $R$ is a bisimulation, which follows from $l^X$ and $l^Y$ being homomorphisms. In detail:

(i) Suppose $sRt$ and $\delta_X(s)(\alpha) \in 1 + \mathrm{At}$. Then $\delta_X(s)(\alpha) = \delta_{\mathcal{L}}(l^X(s))(\alpha) = \delta_{\mathcal{L}}(l^Y(t))(\alpha) = \delta_Y(t)(\alpha)$.

(ii) Suppose $sRt$ and $\delta_X(s)(\alpha) = (\beta, s')$. Then $\delta_{\mathcal{L}}(l^Y(t))(\alpha) = \delta_{\mathcal{L}}(l^X(s))(\alpha) = (\beta, l^X(s'))$. Since $l^Y$ is a homomorphism, the equality $\delta_{\mathcal{L}}(l^Y(t))(\alpha) = (\beta, l^X(s'))$ implies there exists $t'$ for which $\delta_Y(t)(\alpha) = (\beta, t')$ and $(\beta, l^Y(t')) = \delta_{\mathcal{L}}(l^Y(t))(\alpha) = (\beta, l^X(s'))$. Therefore $s'Rt'$. □

We need a lemma which relates the language semantics $[\![e]\!]$ for $e \in \mathbf{GExp}$ with the coalgebraic language acceptance $l^{\tilde{D}}(e)$. The proof of this lemma is in the appendix.

*Lemma 4.9.* For all $e \in \mathbf{GExp}$, the equality $[\![e]\!] = l^{\tilde{D}}(e)$ holds.

We now give the following Hopcroft-Karp style algorithm adapted to $H$-automata. The algorithm uses a linear time finite automata equivalence procedure from [4] using the union-find data structure. The pseudocode for the algorithm is given in Algorithm 1. To formulate the complexity result, we need to define $|e|$. We set $|\mathbf{dup}| = |f \leftarrow n| = |b| \triangleq 1$, $|e +_b g| \triangleq |e| + |g| + 1$, $|e \cdot g| \triangleq |e| + |g|$, and $|e^{(b)}| \triangleq |e| + 1$. Let $\mathbf{GExp}(e)$ be the set of expressions reachable from $e$ under 0 or more applications of $D$ (so $e \in \mathbf{GExp}(e)$). Formally, $\mathbf{GExp}(e) \subseteq \mathbf{GExp}$ is defined as the smallest set of expressions such that

(i) $e \in \mathbf{GExp}(e)$;

(ii) for all $\alpha, \beta \in \mathrm{At}$, if $D(e)(\alpha) = (\beta, e')$ then $e' \in \mathbf{GExp}(e)$;

(iii) whenever $r, s \in \mathbf{GExp}(e)$ and $\alpha, \beta, \gamma \in \mathrm{At}$ satisfy $D(r)(\alpha) = (\beta, s)$ and $D(s)(\beta) = (\gamma, t)$ for some $t \in \mathbf{GExp}$ then $t \in \mathbf{GExp}(e)$.

We mirror the notion of reachability captured in the acceptance condition, which requires the output atom from one transition to be the input atom for the next transition:

$$\begin{aligned} \mathrm{accept}(x, \alpha \cdot \pi_\beta) &\iff \delta_X(x)(\alpha) = \beta \\ \mathrm{accept}(x, \alpha \cdot \pi_\beta \cdot \mathbf{dup} \cdot w) &\iff \exists y.\, \delta_X(x)(\alpha) = (\beta, y) \wedge \mathrm{accept}(y, \beta \cdot w) \end{aligned}$$

We now claim that $|\mathbf{GExp}(e)| \in O(|e|)$. The proof is by induction on $e$.

*Lemma 4.10.* The set $\mathbf{GExp}(e)$ satisfies $|\mathbf{GExp}(e)| \in O(|e|)$.

*Proof.* When $e = f \leftarrow n$ or $e = b$ then $|\mathbf{GExp}(e)| = 1$ and the claim holds. When $e = \mathbf{dup}$ then there are $|\mathrm{At}| + 1 \in O(1)$ elements in $\mathbf{GExp}(e)$. When $e = g +_b h$ then the IH says $|\mathbf{GExp}(g)| \in O(|g|)$ and $|\mathbf{GExp}(h)| \in O(|h|)$. Notice that $\mathbf{GExp}(e) \setminus \{e\} \subseteq \mathbf{GExp}(g) \cup \mathbf{GExp}(h)$, so $|\mathbf{GExp}(e)| \leq |\mathbf{GExp}(g)| + |\mathbf{GExp}(h)| + 1 \in O(|g| + |h| + 1) = O(|e|)$. When $e = g \cdot h$ then the IH says $|\mathbf{GExp}(e)| \in O(|g|)$ and likewise for $h$. Notice that if $D(g \cdot h)(\alpha) = (\beta, r)$ then we have two cases (not necessarily disjoint). Either $r = g' \cdot h$ for some $g \neq g' \in \mathbf{GExp}(g)$ or $r = h'$ for some $h \neq h' \in \mathbf{GExp}(h)$ (the definition of $D$ disallows one-step self-transitions of shape $D(e)(\alpha) = (\beta, e)$). Therefore, one proves by induction on the definition of $\mathbf{GExp}(e)$ that *every* $s \in \mathbf{GExp}(e)$ is of shape $g' \cdot h$ for $g' \in \mathbf{GExp}(g)$ or $h'$ for $h' \in \mathbf{GExp}(h)$ (where now $g' = g$ and $h' = h$ are possible). These cases might not be disjoint if, say, $h = g^{(b)}$, but we only need the bound $|\mathbf{GExp}(e)| \leq |\mathbf{GExp}(g)| + |\mathbf{GExp}(h)| \in O(|g| + |h|) = O(e)$.

Finally, when $e = g^{(b)}$ then the IH says $|\mathbf{GExp}(g)| \in O(|g|)$. Suppose $\beta$ satisfies

$D(g^{(b)})(\beta) = (\gamma, g' \cdot g^{(b)})$. This holds if and only if $\beta \leq b$ and $D(g)(\beta) = (\gamma, g')$. Furthermore it is an easy proof by induction that *every* element of $\mathbf{GExp}(g^{(b)}) \setminus \{g^{(b)}\}$ is of shape $g' \cdot g^{(b)}$ for $g' \in \mathbf{GExp}(g)$. Therefore $|\mathbf{GExp}(g^{(b)})| \leq |\mathbf{GExp}(g)| + 1 \in O(|g| + 1) = O(e)$. The claim $|\mathbf{GExp}(e)| \in O(|e|)$ for all $e$ is proven. □

To decide whether $e, g \in \mathbf{GExp}$ are language-equivalent, we perform the following procedure.

(i) We normalize the finite $H$-coalgebras $(\mathbf{GExp}(e), D)$ and $(\mathbf{GExp}(g), D)$ to get normal finite $H$-automata $(\mathbf{GExp}(e), \tilde{D}, e)$ and $(\mathbf{GExp}(g), \tilde{D}, e)$, from which we automatically get strong normality as sketched towards the end of Section 4.2.

(ii) We tag the states of $\mathbf{GExp}(e)$ and $\mathbf{GExp}(g)$ to enforce disjointness if necessary.

(iii) Then, we feed the automata $(\mathbf{GExp}(e), \tilde{D}, e)$ and $(\mathbf{GExp}(g), \tilde{D}, g)$ into Algorithm 1 and output **true** if $e \sim g$, **false** otherwise.

**Theorem 4.11.** The above procedure correctly decides whether $[\![e]\!] = [\![g]\!]$ in time $O(n \cdot \alpha(n))$ for $|\mathrm{At}|$ constant, where $\alpha$ is the inverse Ackermann function and $n = |e| + |g|$.

*Proof.* To see correctness, by Lemma 4.9 it holds that $[\![e]\!] = [\![g]\!]$ if and only if $l^{\tilde{D}}(e) = l^{\tilde{D}}(g)$. By strong normality of $(\mathbf{GExp}, \tilde{D})$, Corollary 4.8 says $l^{\tilde{D}}(e) = l^{\tilde{D}}(g)$ if and only if $e \sim g$. Then, correctness of the algorithm follows by correctness of Hopcroft-Karp [4].

To see the time complexity result, recall that $|\mathbf{GExp}(e)| \in O(|e|)$ and $|\mathbf{GExp}(g)| \in O(|g|)$. Furthermore to pre-compute these sets takes at most time

Algorithm 1: Hopcroft-Karp algorithm for equivalence of $H$-automata [4]

**Input:** $\mathcal{X} = (X, \delta_X, i_X)$ and $\mathcal{Y} = (Y, \delta_Y, i_Y)$ finite disjoint strongly normal $H$-coalgebras

**Output:** **true** if $\mathcal{X}$ and $\mathcal{Y}$ are equivalent, **false** otherwise

todo ← Queue.singleton($i_X, i_Y$);
forest ← UnionFind.disjointForest($X \uplus Y$);
**while not** todo.isEmpty() **do**
    $x, y \leftarrow$ todo.pop();
    $r_x, r_y \leftarrow$ forest.find($x$), forest.find($y$);
    **if** $r_x = r_y$ **then continue**;
    **end if**
    **for** $\alpha \in$ At **do**
        **switch** $\delta_X(x)(\alpha), \delta_Y(y)(\alpha)$ **do**
            **case** $b_1, b_2$ with $b_1 = b_2$: **continue**;
            **case** $(\beta, x'), (\beta, y')$: todo.push($x', y'$);
            **case** _, _: **return false**;
    **end for**
    forest.union($r_x, r_y$);
**end while**
**return true**;

$|\text{At}| \cdot |\mathbf{GExp}(e)| \in O(|e|)$ and $|\text{At}| \cdot |\mathbf{GExp}(g)| \in O(|g|)$ respectively, since computing the states $\mathbf{GExp}(e)$ amounts to following all traces which begin at $e$ in the coalgebra $(\mathbf{GExp}, D)$ and proceeding outwards, enforcing the rule that any output atom is the input atom for the next transition. The tagging of $\mathbf{GExp}(e)$ and $\mathbf{GExp}(g)$ to enforce disjointness can be done as we compute each respectively.

To normalize the coalgebras $(\mathbf{GExp}(e), D)$ and $(\mathbf{GExp}(g), D)$ to $(\mathbf{GExp}(e), \tilde{D})$ and $(\mathbf{GExp}(g), \tilde{D})$ respectively, we perform a reverse breadth-first search starting from *accepting* states, defined as those $s \in \mathbf{GExp}(e)$ (resp. $g$) for which there exists $\alpha, \beta \in \mathrm{At}$ satisfying $D(s)(\alpha) = \beta$. We mark all accepting states as live, and from each accepting state we traverse (in a breadth-first manner) the labelled directed graph arising from $(\mathbf{GExp}(e), D)$, marking each state as we reach them. Recall that edges are included in $\mathbf{GExp}(e)$ only if they respect the output-as-next-input scheme starting from $e$, by definition. A state will then remain unmarked if and only if it is dead, since all live states will be reached and marked according to whatever trace witnessed their liveness. This reverse breadth-first search takes time $O(|e|)$ since there are $O(|e|)$ many states and at most $|\mathrm{At}| \in O(1)$ many transitions out of each state, and likewise for $g$.

Finally, since $(\mathbf{GExp}(e), \tilde{D}, e)$ has $O(|e|)$ many states and at most $|\mathrm{At}| \in O(1)$ transitions out of each state (and likewise for $g$), by a classic result due to Robert Tarjan [12] establishes that Hopcroft-Karp takes time $O(n \cdot \alpha(n))$ to terminate, where $n = |e| + |g|$ [11]. □

# CHAPTER 5

# APPLICATIONS AND CONCLUDING REMARKS

We arrive in hand with soundness and partial-completeness for GNetKAT along with a nearly linear semantic equivalence decision procedure. The original NetKAT framework has numerous concrete applications of its decision procedure, which include but are not limited to:

(i) deciding *reachability* between hosts, answering the question "Can host $A$ send packets (along some path) to host $B$?";

(ii) deciding whether a location $w$ is a *waypoint* relative to two other locations $a$ and $b$, answering the question "Does every packet traveling from host $A$ to host $B$ need to pass through host $W$?";

(iii) deciding whether traffic through a portion of the network is *isolated*, meaning that certain locations' packets cannot be processed by other locations.

Each of these is encoded in a relatively straightforward manner into the NetKAT framework in [1]. Endpoints for a network in this context are modeled as Boolean predicates. For example, the authors in [1] decide (i) by modeling the endpoints as predicates $a$ and $b$ and first defining "$b$ is *reachable* from $a$" as equivalent to there existing a trace $\langle pk_1, \ldots, pk_n \rangle \in \bigcup \text{range} [\![ \mathbf{dup} \cdot (p \cdot t \cdot \mathbf{dup})^* ]\!]_H$ for which $[\![a]\!]_H(\langle pk_n \rangle) = \{\langle pk_n \rangle\}$ and $[\![b]\!]_H(\langle pk_1 \rangle) = \{\langle pk_1 \rangle\}$. Then, by their previous results this is equivalent to checking whether $a \cdot \mathbf{dup} \cdot (p \cdot t \cdot \mathbf{dup})^* \cdot b \not\equiv 0$, where $p$ encodes the behavior of the switches in the network (e.g. access control) and $t$ encodes the topology of the network as in Section 1.1.

Due to the more syntactically restrictive nature of GNetKAT and the lack of a full completeness theorem, the difficulty in adapting the NetKAT decision

problem encodings to GNetKAT encodings ranges from modest to very difficult. In particular, the solution to point (ii) in the original NetKAT setting makes use of the natural partial order $\leq$ which arises from any Kleene Algebra $K$ by $x \leq y \iff x + y = y$ for $x, y \in K$. Unfortunately, there does not seem to be an analogous canonical ordering on a given GKAT, since the + of KA is replaced by $+_b$ for $b \in \mathbf{BExp}$ in GKAT as remarked in [9]. Therefore another approach entirely needs to be taken to analyze waypoint correctness as originally defined in [1]. Furthermore, the analysis of traffic isolation for (iii) as in NetKAT is phrased in terms of unions (nondeterministic choices) between subpolicies. It is not clear how to effectively model subnetworks which may interfere with one another after having processed some amount of state (determinism).

We demonstrate the GNetKAT semantic equivalence procedure by deciding (i) analogously to NetKAT and by deciding (ii) in a novel way, since these approaches are more tractable for our setting. First, we need to describe more of the network programming model as given in [1]. Recall the forwarding policy from Section 1.1

$$p \triangleq (dst = H_1 \cdot pt \leftarrow 1) + (dst = H_2 \cdot pt \leftarrow 2)$$

which is the union of two policies which respectively set the $pt$ field to $i$ if a packet is destined for location $H_i$. Note that $dst$ and $pt$ are field names, and $H_i$ is some integer encoding of the location. Unfortunately, $p$ is a KAT term and not a GKAT term, but we can transform it to a GKAT term $p_G$ rather easily. The key fact of relevance is the universally quantified equation $f = n \cdot \neg(f = m) \equiv f = n$ for a field $f$ and (integer) values $n \neq m$, which is provable just from the KAT and packet algebra axioms, see Example 2.7. Using the KAT and packet algebra

axioms gives

$$\begin{aligned} p &\triangleq (dst = H_1 \cdot pt \leftarrow 1) + (dst = H_2 \cdot pt \leftarrow 2) \\ &\equiv dst = H_1 \cdot pt \leftarrow 1 + \neg(dst = H_1) \cdot dst = H_2 \cdot pt \leftarrow 2 \end{aligned}$$

Therefore $p$ is provably equivalent in NetKAT to the homomorphic image of the GNetKAT term $pt \leftarrow 1 +_{dst=H_1} (dst = H_2 \cdot pt \leftarrow 2)$ under the standard homomorphism from GNetKAT into NetKAT as described in Section 3.2, analogous to abstract GKAT to abstract KAT in [11]. We can therefore soundly encode the forwarding policy $p$ in GNetKAT as $p_G \triangleq pt \leftarrow 1 +_{dst=H_1} (dst = H_2 \cdot pt \leftarrow 2)$.

This demonstrates a common method that we will use to transform unions of NetKAT terms to guarded unions of GNetKAT terms: aggressively apply the sound, universally quantified fact $f = n \equiv \neg(f = m) \cdot f = n$ (when $n \neq m$) in order to obtain the shape of a guarded sum. As a brief sketch on how to transform loops (which we demonstrate later), we choose some suitable Boolean expression to test against based on the network and task at hand, and we attempt to rewrite nondeterministic star expressions in terms of deterministic while-loops.

The NetKAT paper also defines the essential notion of the *logical crossbar* model, which encodes the end-to-end behavior of a network with: a pair of input/output predicates named *in* and *out* respectively, a **dup**-free network policy $p$ encoding the behavior of switches (e.g. access control, forwarding), and a **dup**-free network topology $t$ which encodes valid connections between hosts. The NetKAT term modeling the end-to-end behavior is then $in \cdot (p \cdot t)^* \cdot out$. Intuitively, we filter for input/output and repeatedly attempt to process packets according to the network policy $p$ and topology $t$. This model can be generalized further to $in \cdot (p \cdot t \cdot \mathbf{dup})^* \cdot out$ where $p$ and $t$ are **dup**-free terms, allowing us to

record the individual "hops" that a given packet may take in the network. The generalized logical crossbar model is ubiquitous in the concrete applications of the NetKAT framework [1].

## 5.1 Reachability

We would like to formulate a method to decide reachability in the GNetKAT setting. We follow the methods employed by the authors in the original NetKAT paper, adapting proofs to their guarded variants. Throughout this section we assume that network policies and encoded topologies are expressed in their reduced form under the sound transformation $\mathbf{GExp} \to \mathbf{GExp}^-$ described in Section 3.3.1.

*Definition 5.1.* For $a, b \in \mathbf{BExp}$ and $p, t \in \mathbf{GExp}^-$ which are **dup**-free, we say that "$b$ is *reachable* from $a$ (relative to policy $p$ and topology $t$)" if and only if there exists some trace $\langle pk_1, \ldots, pk_n \rangle \in \bigcup \text{range}(\llbracket a \cdot \mathbf{dup} \cdot (p \cdot t \cdot \mathbf{dup})^{(\overline{b})} \rrbracket_H)$, equivalently that there is some packet history $h$ satisfying $\llbracket a \cdot \mathbf{dup} \cdot (p \cdot t \cdot \mathbf{dup})^{(\overline{b})} \rrbracket_H(h) \neq \emptyset$.

Notice that an equivalent definition of "$b$ is reachable from $a$" is the proposition $\llbracket 0 \rrbracket_H \neq \llbracket a \cdot \mathbf{dup} \cdot (p \cdot t \cdot \mathbf{dup})^{(\overline{b})} \rrbracket_H$ as functions $H \to \mathcal{P}(H)$. Since we assumed $p, t \in \mathbf{GExp}^-$, we can write $\overline{b} \equiv_{\mathbf{BExp}} \vee_{\alpha \in E} \alpha$ for some $E \subseteq \text{At}$ so that $a \cdot \mathbf{dup} \cdot (p \cdot t \cdot \mathbf{dup})^{(E)} \in \mathbf{GExp}^-$. Then, Lemma 2.5 says

$$\llbracket a \cdot \mathbf{dup} \cdot (p \cdot t \cdot \mathbf{dup})^{(E)} \rrbracket_H \neq \llbracket 0 \rrbracket_H \iff \llbracket a \cdot \mathbf{dup} \cdot (p \cdot t \cdot \mathbf{dup})^{(E)} \rrbracket \neq \llbracket 0 \rrbracket$$

Therefore, to decide whether $b$ is reachable from $a$ relative to policy $p$ and

topology $t$, it suffices to decide using the decision procedure from Chapter 4 whether $a \cdot \mathbf{dup} \cdot (p \cdot t \cdot \mathbf{dup})^{(E)} \nsim 0$, where the former is the start state of the automaton with coalgebra $\left(\mathbf{GExp}\left(a \cdot \mathbf{dup} \cdot (p \cdot t \cdot \mathbf{dup})^{(E)}\right), \tilde{D}\right)$ and the latter is the start state of the automaton with coalgebra $(\{0\}, \tilde{D})$. By Theorem 4.11, this procedure is correctly performed in time $O(n \cdot \alpha(n))$ with $n = |p| + |t| + 5$, since $\left|a \cdot \mathbf{dup} \cdot (p \cdot t \cdot \mathbf{dup})^{(E)}\right| = |p| + |t| + 4$ and $|0| = 1$.

## 5.2 Waypoints

As mentioned earlier in this chapter, the NetKAT approach to deciding whether a location $w$ is a waypoint relative to locations $a$ and $b$ involves exploiting the natural partial order on any KA [1]. We cannot imitate their method in this setting, so we propose an alternative way to decide whether $w$ is a waypoint for $a$ and $b$. We again assume that network policies $p$ and topologies $t$ are in their reduced form.

*Definition 5.2 (Waypoint).* A predicate $w$ is a *waypoint* for predicates (locations) $a$ and $b$ (relative to **dup**-free policy $p$ and **dup**-free topology $t$) if and only if for every history $\langle pk_1, \ldots, pk_n \rangle \in \bigcup \text{range}([\![a \cdot \mathbf{dup} \cdot (p \cdot t \cdot \mathbf{dup})^{(\bar{b})}]\!]_H)$, there exists some $pk_x \in \{pk_1, \ldots, pk_n\}$ for which

(i) $[\![w]\!]_H(\langle pk_x \rangle) = \{\langle pk_x \rangle\}$;

(ii) $[\![b]\!]_H(\langle pk_i \rangle) = \emptyset$ for all $1 < i < x$;

(iii) $[\![a]\!]_H(\langle pk_i \rangle) = \emptyset$ for all $x < i < n$.

Given $a, b, w$ and $p, t$ as in Definition 5.2, consider the GNetKAT term

$$a \cdot \mathbf{dup} \cdot (\neg b \cdot p \cdot t \cdot \mathbf{dup})^{(\overline{w})} \cdot (\neg a \cdot p \cdot t \cdot \mathbf{dup})^{(\overline{b})}$$

Intuitively, this term encodes the same hop-by-hop behavior to get from $a$ to $b$, but it also enforces that $b$ never holds until $w$ does and $a$ never holds after $w$ does. Therefore, if $w$ truly is a waypoint for $a$ and $b$ relative to $p$ and $t$, then the equation

$$[\![a \cdot \mathbf{dup} \cdot (p \cdot t \cdot \mathbf{dup})^{(\overline{b})}]\!]_H = [\![a \cdot \mathbf{dup} \cdot (\neg b \cdot p \cdot t \cdot \mathbf{dup})^{(\overline{w})} \cdot (\neg a \cdot p \cdot t \cdot \mathbf{dup})^{(\overline{b})}]\!]_H$$

holds by definition of $w$ being a waypoint. Conversely, if the above equation holds, then for every starting history the respective image sets are equal, so in particular every element $\langle pk_1, \ldots, pk_n \rangle \in \bigcup \text{range}([\![a \cdot \mathbf{dup} \cdot (p \cdot t \cdot \mathbf{dup})^{(\overline{b})}]\!]_H)$ is also an element of $\bigcup \text{range}([\![a \cdot \mathbf{dup} \cdot (\neg b \cdot p \cdot t \cdot \mathbf{dup})^{(\overline{w})} \cdot (\neg a \cdot p \cdot t \cdot \mathbf{dup})^{(\overline{b})}]\!]_H)$, from which we infer that there must exist some $pk_x \in \{pk_1, \ldots, pk_n\}$ satisfying

(i) $[\![w]\!]_H(\langle pk_x \rangle) = \{\langle pk_x \rangle\}$;

(ii) $[\![b]\!]_H(\langle pk_i \rangle) = \emptyset$ for all $1 < i < x$;

(iii) $[\![a]\!]_H(\langle pk_i \rangle) = \emptyset$ for all $x < i < n$.

Since this holds for every element $\langle pk_1, \ldots, pk_n \rangle \in \bigcup \text{range}([\![a \cdot \mathbf{dup} \cdot (p \cdot t \cdot \mathbf{dup})^{(\overline{b})}]\!]_H)$, it follows by definition that $w$ is a waypoint.

Now, let $E_w \triangleq \{\alpha \in \text{At} : \alpha \leq \overline{w}\}$ and likewise $E_b \triangleq \{\alpha \in \text{At} : \alpha \leq \overline{b}\}$, so $\overline{w} \equiv_{\mathbf{BExp}} \vee_{\alpha \in E_w} \alpha$ and $\overline{b} \equiv \vee_{\alpha \in E_b} \alpha$. Recall that we assumed $p$ and $t$ were reduced,

so we are in a situation to apply Lemma 2.5 and our algorithm. Specifically, Lemma 2.5 says

$$\begin{aligned} & [\![a \cdot \mathbf{dup} \cdot (p \cdot t \cdot \mathbf{dup})^{(E_b)}]\!]_H = [\![a \cdot \mathbf{dup} \cdot (\neg b \cdot p \cdot t \cdot \mathbf{dup})^{(E_w)} \cdot (\neg a \cdot p \cdot t \cdot \mathbf{dup})^{(E_b)}]\!]_H \\ \iff \quad & [\![a \cdot \mathbf{dup} \cdot (p \cdot t \cdot \mathbf{dup})^{(E_b)}]\!] = [\![a \cdot \mathbf{dup} \cdot (\neg b \cdot p \cdot t \cdot \mathbf{dup})^{(E_w)} \cdot (\neg a \cdot p \cdot t \cdot \mathbf{dup})^{(E_b)}]\!] \end{aligned}$$

Therefore, letting $n = 3|p| + 3|t| + 12$ be the sum of the sizes of the two expressions above, we have by Theorem 4.11 that we can decide the above equation (whether $w$ is a waypoint) in time $O(n \cdot \alpha(n))$.

## 5.3 Concluding Remarks

We have seen two concrete examples of deciding network properties in the GNetKAT framework analogous to the original NetKAT paper [1]. In each case, we were able to leverage the semantic equivalence procedure from Chapter 4 to decide, in time nearly linear in the size of the network, various network properties of interest like reachability, waypoints, etc. In this manner, we demonstrate how one would emulate common KAT proofs and algorithms in a guarded setting, while also highlighting the inherent challenges involved in restricting the syntax of the language. There is much work yet to be done pertaining to the axiomatization of abstract GKAT and GNetKAT, in particular towards better understanding the role of the non-algebraic fixpoint axiom from GKAT as it relates to soundness and completeness.

We now conclude and reflect upon what we have learned. While GNetKAT is trivially a GKAT as defined in [11], it is deceptively difficult to adapt the

uninterpreted equational theory from GKAT to an interpreted equational theory. Certain axioms which one would expect to be sound are sound no longer, and various attempts to sidestep the issue do not seem to bear fruit. Instead, we appear to be circling around a subtler issue entirely, that being that relating uninterpreted, free theories to those theories with extra commutativity conditions requires deep insight into the effects of the commutativity conditions on the underlying semantics. Furthermore the type (shape) of the semantics has a surprising impact on analysis of soundness, in particular passing from GKAT's language model of type $\mathrm{At} \cdot (\Sigma \cdot \mathrm{At})^*$ to GNetKAT's (and NetKAT's) language model of type $\mathrm{At} \cdot \Pi \cdot (\mathbf{dup} \cdot \Pi)^*$ greatly complicates the empty word property, as discussed in Section 3.1.

There have been attempts to simplify the translation from uninterpreted KAT theories to interpreted ones, with one framework in particular, Kleene Algebra modulo Theories (KMT) [3], purporting to have derived a uniform way to specify primitive (equational) theories, instantiate KAT to those theories, and automatically derive an accompanying decision procedure for the extended theory. Unfortunately, the arguments presented in [3] are complex and rely upon a system of inference rules which are *not* easily seen to be well-founded, and further attempts to clarify the arguments in [3] have not settled the issue. It would be highly desirable to have a well-understood *guarded* Kleene Algebra modulo Theories, which would provide a uniform way to argue soundness, completeness, and decidability for a whole class of equational theories of interest, naturally subsuming the GNetKAT case. This remains open as a potential direction of research in the field of formal methods and program verification. In addition, we leave open how to properly axiomatize GNetKAT with loops and how to prove completeness in the presence of such axiom(s).

APPENDIX A

**OMITTED PROOFS**

## A.1 Chapter 2

*Proof of Lemma 2.3.* By induction on $e \in \mathbf{GExp}^{-}$.

Case: $e = \alpha$. Then $[\![\alpha]\!] = \{\alpha \cdot \pi_\alpha\}$ and $[\![\alpha]\!]_H(pk :: l) = \begin{cases} \{pk :: l\} & \alpha \sim pk \\ \emptyset & \text{otherwise} \end{cases}$ where we write $\alpha \sim pk$ to indicate $\alpha$ and $pk$ agree on all field values. Therefore our desired equality boils down to $[\![\alpha]\!]_H(h) = [\![\alpha \cdot \pi_\alpha]\!]_H(h)$ for all $h$. This is easily seen by case analysis letting $h = pk :: l$: if $\alpha \sim pk$ then $[\![\alpha]\!]_H(h) = \{h\} = ([\![\pi_\alpha]\!]_H \ominus [\![\alpha]\!]_H)(h)$, otherwise if $\alpha \nsim pk$ then $[\![\alpha]\!]_H(h) = \emptyset = [\![\alpha \cdot \pi_\alpha]\!]_H(h)$, so this case holds.

Case: $e = \pi$. Then $[\![\pi]\!] = \{\alpha \cdot \pi : \alpha \in \mathrm{At}\}$ and $[\![\pi]\!]_H(pk :: h) = \{pk' :: h\}$ where $\alpha_\pi \sim pk'$ using the notation from the previous case. Our desired equality is $[\![\pi]\!]_H(h) = \bigcup_{\alpha \in \mathrm{At}} [\![\alpha \cdot \pi]\!]_H(h)$ for all $h$. This is seen as follows: letting $h = pk :: l$, there exists exactly one $\beta \in \mathrm{At}$ for which $\beta \sim pk$. Therefore $\bigcup_{\alpha \in \mathrm{At}} [\![\alpha \cdot \pi]\!]_H(h) = [\![\beta \cdot \pi]\!]_H(h) = [\![\pi]\!]_H(h)$, since $\beta \sim pk$ so $[\![\beta]\!]_H(h) = \{h\}$. This case holds.

Case: $e = g +_b f$. By the IH, we know $[\![g]\!]_H(h) = \bigcup_{x \in [\![g]\!]} [\![x]\!]_H(h)$ for all $h$ and $[\![f]\!]_H(h) = \bigcup_{x \in [\![f]\!]} [\![f]\!]_H(h)$ for all $h$. Our desired equality is

$$[\![g +_b f]\!]_H(h) = \bigcup_{x \in [\![g +_b f]\!]} [\![x]\!]_H(h) \qquad \forall h \in H$$

Recall that $x \in [\![g +_b f]\!] \iff x \in [\![b]\!] \diamond [\![g]\!]$ or $x \in [\![\bar{b}]\!] \diamond [\![f]\!]$. It is useful then to write

$$\bigcup_{x \in [\![g +_b f]\!]} [\![x]\!]_H(h) = \left( \bigcup_{x \in [\![b]\!] \diamond [\![g]\!]} [\![x]\!]_H(h) \right) \cup \left( \bigcup_{x \in [\![\overline{b}]\!] \diamond [\![f]\!]} [\![x]\!]_H(h) \right)$$

for the following reason: let $h \in H$ and write $h = pk :: l$, and introduce the notation $pk \leq b$ for $b \in \mathbf{BExp}$, $pk$ a packet if $[\![b]\!]_H(pk :: l) = \{pk :: l\}$. We proceed by case analysis on whether $pk \leq b$ holds.

Subcase: $pk \leq b$ holds. Note that $pk \leq b$ holds if and only if $\rho \cdot \pi_\rho \in [\![b]\!]$ where $\rho$ is the unique atom satisfying $\rho \sim pk$. In this case, then $[\![g +_b f]\!]_H(h) = [\![g]\!]_H(h)$. By the IH, $[\![g]\!]_H(h) = \bigcup_{x \in [\![g]\!]} [\![x]\!]_H(h)$. Note that, since $\rho \cdot \pi_\rho \in [\![b]\!]$ we know $\rho \cdot \pi_\rho \notin [\![\overline{b}]\!]$ and hence

$$[\![g +_b f]\!]_H(h) = [\![g]\!]_H(h) = \bigcup_{x \in [\![g]\!]} [\![x]\!]_H(h) = \bigcup_{x \in [\![g +_b f]\!]} [\![x]\!]_H(h)$$

so the statement holds in this case.

Subcase: $pk \not\leq b$ holds. Then, for $\rho$ the unique atom satisfying $\rho \sim pk$, we know $\rho \cdot \pi_\rho \notin [\![b]\!]$ which implies $\rho \cdot \pi_\rho \in [\![\overline{b}]\!]$. Furthermore $pk \not\leq b \iff pk \leq \overline{b}$, so $[\![g +_b f]\!]_H(h) = [\![f]\!]_H(h)$. By the IH, $[\![f]\!]_H(h) = \bigcup_{x \in [\![f]\!]} [\![x]\!]_H(h)$. Since $\rho \not\leq b$ and $\rho \leq \overline{b}$, we know that from the decomposition above that $\bigcup_{x \in [\![g +_b f]\!]} [\![x]\!]_H(h) = \bigcup_{x \in [\![f]\!]} [\![x]\!]_H(h)$. Therefore we have shown the equality $[\![g +_b f]\!]_H(h) = \bigcup_{x \in [\![g +_b f]\!]} [\![x]\!]_H(h)$ so the statement holds in this case.

Case: $e = g \cdot f$. Our desired equality is that $[\![g \cdot f]\!]_H(h) = \bigcup_{x \in [\![g \cdot f]\!]} [\![x]\!]_H(h)$ for all $h$. Note that $x \in [\![g \cdot f]\!] \iff x = y \diamond z$ for $y \in [\![g]\!], z \in [\![f]\!]$. Recall $[\![g \cdot f]\!]_H(h) = ([\![f]\!]_H \ominus [\![g]\!]_H)(h)$ is Kleisli composition. That is,

$$(\llbracket f \rrbracket_H \ominus \llbracket g \rrbracket_H)(h) = \bigcup_{h' \in \llbracket g \rrbracket_H(h)} \llbracket f \rrbracket_H(h')$$

Let $w \in \llbracket g \cdot f \rrbracket_H(h)$ so that there exists $h' \in \llbracket g \rrbracket_H(h)$ for which $w \in \llbracket f \rrbracket_H(h')$. The IH says $\llbracket g \rrbracket_H(h) = \bigcup_{x \in \llbracket g \rrbracket} \llbracket x \rrbracket_H(h)$, so $h' \in \llbracket g \rrbracket_H(h)$ if and only if there exists $x \in \llbracket g \rrbracket$ for which $h' \in \llbracket x \rrbracket_H(h)$. Now, we know $w \in \llbracket f \rrbracket_H(h')$, so by the IH there must exist $y \in \llbracket f \rrbracket$ for which $w \in \llbracket y \rrbracket_H(h')$. It follows that $x \diamond y \in \llbracket g \cdot f \rrbracket$ and that $w \in (\llbracket y \rrbracket_H \ominus \llbracket x \rrbracket_H)(h) = \llbracket x \cdot y \rrbracket_H(h)$. We have therefore shown one direction of the inclusion.

For the reverse inclusion, let $w \in \bigcup_{x \in \llbracket g \cdot f \rrbracket} \llbracket x \rrbracket_H(h)$. Let $x = y \diamond z$ witness $w \in \llbracket y \diamond z \rrbracket_H(h)$. We know that $y \in \llbracket g \rrbracket$ and $z \in \llbracket f \rrbracket$, so by the IH we know that $\llbracket y \rrbracket_H(s) \subseteq \llbracket g \rrbracket_H(s)$ for all $s$ and likewise $\llbracket z \rrbracket_H(s) \subseteq \llbracket f \rrbracket_H(s)$ for all $s$. Now, note that the existence of $y \diamond z$ implies $\llbracket y \diamond z \rrbracket_H(h) = \llbracket y \cdot z \rrbracket_H(h)$, meaning $w \in \llbracket y \cdot z \rrbracket_H(h) = (\llbracket z \rrbracket_H \ominus \llbracket y \rrbracket_H)(h)$. Therefore, there exists $h' \in \llbracket y \rrbracket_H(h)$ for which $w \in \llbracket z \rrbracket_H(h')$ holds. Using the subset containments previously mentioned, $h' \in \llbracket y \rrbracket_H(h) \subseteq \llbracket g \rrbracket_H(h)$ and $w \in \llbracket z \rrbracket_H(h') \subseteq \llbracket f \rrbracket_H(h')$ both hold. Therefore we have shown

$$\llbracket g \cdot f \rrbracket_H(h) = (\llbracket f \rrbracket_H \ominus \llbracket g \rrbracket_H)(h) \supseteq \llbracket y \diamond z \rrbracket_H(h) = \llbracket y \cdot z \rrbracket_H(h) \ni w$$

so the reverse inclusion holds as well. We are done.

Case: $e = \mathbf{dup}$. Then $\llbracket \mathbf{dup} \rrbracket_H(pk :: l) = \{pk :: (pk :: l)\}$ and $\llbracket \mathbf{dup} \rrbracket = \{\alpha \cdot \pi_\alpha \cdot \mathbf{dup} \cdot \pi_\alpha : \alpha \in \mathrm{At}\}$. We wish to show that

$$\llbracket \mathbf{dup} \rrbracket_H(pk :: l) = \bigcup_{\alpha \in \mathrm{At}} \llbracket \alpha \cdot \pi_\alpha \cdot \mathbf{dup} \cdot \pi_\alpha \rrbracket_H(pk :: l)$$

for every $pk$, $l$ a list of packets (possibly empty). Let $h = pk :: l \in H$. Let $\rho \in \mathrm{At}$ be the unique atom for which $\rho \sim pk$. Then clearly

$$\bigcup_{\alpha \in \mathrm{At}} [\![\alpha \cdot \pi_\alpha \cdot \mathbf{dup} \cdot \pi_\alpha]\!]_H(pk :: l) = [\![\rho \cdot \pi_\rho \cdot \mathbf{dup} \cdot \pi_\rho]\!]_H(pk :: l)$$

since any $\beta \in \mathrm{At}$ different from $\rho$ will drop the packet. Since $\rho \sim pk$ it follows directly that $[\![\rho \cdot \pi_\rho \cdot \mathbf{dup} \cdot \pi_\rho]\!]_H(pk :: l) = \{pk :: (pk :: l)\} = [\![\mathbf{dup}]\!]_H(pk :: l)$. We are done in this case.

Case: $e = g^{(A)}$. We know that $[\![g^{(A)}]\!]_H(h) = \bigcup_{n \in \mathbb{N}} ([\![\overline{A}]\!]_H \ominus G^n)(h)$ where $G^0(h) = \{h\}, G^{n+1}(h) = ([\![A \cdot g]\!]_H \ominus G^n)(h)$, $\overline{A} = \mathrm{At} \setminus A$. Note that we are treating $A \subseteq \mathrm{At}$ as a disjunction of Booleans as far as the packet history semantics are concerned. We also know that

$$[\![g^{(A)}]\!] = \left( \bigcup_{n \in \mathbb{N}} ([\![A]\!] \diamond [\![g]\!])^{\underline{n}} \right) \diamond [\![\overline{A}]\!]$$

$$= \bigcup_{n \in \mathbb{N}} \left( ([\![A]\!] \diamond [\![g]\!])^{\underline{n}} \diamond [\![\overline{A}]\!] \right)$$

We need to show that $[\![g^{(A)}]\!]_H(h) = \bigcup_{x \in [\![g^{(A)}]\!]} [\![x]\!]_H(h)$ for all $h$. The IH tells us that $[\![g]\!]_H(h) = \bigcup_{x \in [\![g]\!]} [\![x]\!]_H(h)$ for all $h$. Notice that $x \in [\![g^{(A)}]\!] \iff$ there exists $n \geq 0$ for which $x \in ([\![A]\!] \diamond [\![g]\!])^{\underline{n}} \diamond [\![\overline{A}]\!]$.

If this $n \geq 0$ exists then it is necessarily unique: the claim is obvious for $n = 0$ since $x \in [\![\overline{A}]\!]$. If $0 \leq k < n$ and $x \in ([\![A]\!] \diamond [\![g]\!])^{\underline{k}} \diamond [\![\overline{A}]\!]$, then $x$ factors like $x = y \diamond z$ for $y \in ([\![A]\!] \diamond [\![g]\!])^{\underline{k}}$ and $z \in [\![\overline{A}]\!]$. Therefore the trailing complete assignment $\pi_\beta$ of $y$ satisfies $\beta \notin A$. However, since

$$x \in (\llbracket A \rrbracket \diamond \llbracket g \rrbracket)^{\underline{n}} = (\llbracket A \rrbracket \diamond \llbracket g \rrbracket)^{\underline{k}} \diamond (\llbracket A \rrbracket \diamond \llbracket g \rrbracket)^{\underline{n-k}}$$

So let $x = y \diamond w$ for $y \in (\llbracket A \rrbracket \diamond \llbracket g \rrbracket)^{\underline{k}}, w \in (\llbracket A \rrbracket \diamond \llbracket g \rrbracket)^{\underline{n-k}}$. Note that we intentionally reuse the variable $y$ since determinacy of languages enforces that $y \in (\llbracket A \rrbracket \diamond \llbracket g \rrbracket)^{\underline{k}}$ (which was given as an assumption earlier) is the unique string prefixed by $\alpha$ in $(\llbracket A \rrbracket \diamond \llbracket g \rrbracket)^{\underline{k}}$. Furthermore $w \in (\llbracket A \rrbracket \diamond \llbracket g \rrbracket)^{\underline{n-k}}$ where $n - k > 0$ forces the leading atom in $w$, say $\delta$, to satisfy $\delta \in A$. Therefore, recalling that the trailing complete assignment in $y$ was $\pi_\beta$ where $\beta \notin A$, we have a contradiction: $y \diamond w$ cannot be defined. A very similar argument carries through when $k > n$: use the decomposition $k = n + (k - n)$ with the factorizations witnessed by $x \in (\llbracket A \rrbracket \diamond \llbracket g \rrbracket)^{\underline{n}} \diamond \llbracket \overline{A} \rrbracket$ and $x \in (\llbracket A \rrbracket \diamond \llbracket g \rrbracket)^{\underline{k}} \diamond \llbracket \overline{A} \rrbracket$ to derive a contradiction using determinacy (since $k - n > 0$). Therefore

$$x \in \llbracket g^{(A)} \rrbracket \iff \exists! n \in \mathbb{N}.\, x \in (\llbracket A \rrbracket \diamond \llbracket g \rrbracket)^{\underline{n}} \diamond \llbracket \overline{A} \rrbracket$$

We now proceed with the rest of the claim. We first show by induction that $x = y_1 \diamond \ldots \diamond y_n \in (\llbracket A \rrbracket \diamond \llbracket g \rrbracket)^{\underline{n}} \diamond \llbracket \overline{A} \rrbracket$ implies $\llbracket x \rrbracket_H(h) \subseteq (\llbracket \overline{A} \rrbracket_H \ominus G^n)(h)$ for all $h$. When $n = 0$ then $x \in \llbracket \overline{A} \rrbracket$ and the claim is trivial. When $n = n' + 1$, apply the IH to the string $y_2 \diamond \ldots \diamond y_{n'+1} \in (\llbracket A \rrbracket \diamond \llbracket g \rrbracket)^{\underline{n'}} \diamond \llbracket \overline{A} \rrbracket$ to find

$$\llbracket y_2 \diamond \ldots \diamond y_{n'+1} \rrbracket_H(h) = \llbracket y_2 \cdot \ldots \cdot y_{n'+1} \rrbracket_H(h) \subseteq (\llbracket \overline{A} \rrbracket_H \ominus G^{n'})(h)$$

for all $h$. So since $y_1 \in \llbracket A \rrbracket \diamond \llbracket g \rrbracket \subseteq \llbracket g \rrbracket$ and $g$ satisfies the IH, we conclude

$$\llbracket y_1 \diamond y_2 \diamond \ldots \diamond y_{n'+1} \rrbracket_H(h) = \llbracket y_1 \cdot y_2 \cdot \ldots \cdot y_{n'+1} \rrbracket_H(h)$$

$$\subseteq (\llbracket \overline{A} \rrbracket_H \ominus G^{n'} \ominus \llbracket g \rrbracket_H \ominus \llbracket A \rrbracket_H)(h) = (\llbracket \overline{A} \rrbracket_H \ominus G^{n'+1})(h)$$

Therefore we have shown $x \in (\llbracket A \rrbracket \diamond \llbracket g \rrbracket)^{\underline{n}} \diamond \llbracket \overline{A} \rrbracket$ implies $\llbracket x \rrbracket_H(h) \subseteq (\llbracket \overline{A} \rrbracket_H \ominus G^n)(h)$ for all $n$ and for all $h$. We continue like

$$\begin{aligned}
& \bigcup_{x \in \llbracket g^{(A)} \rrbracket} \llbracket x \rrbracket_H(h) \\
= \; & \bigcup_{n \geq 0} \bigcup \{ \llbracket x \rrbracket_H(h) : x \in (\llbracket A \rrbracket \diamond \llbracket g \rrbracket)^{\underline{n}} \diamond \llbracket \overline{A} \rrbracket \} && \text{(def.)} \\
\subseteq \; & \bigcup_{n \geq 0} (\llbracket \overline{A} \rrbracket_H \ominus G^n)(h) && \text{(from above)} \\
= \; & \llbracket g^{(A)} \rrbracket_H(h) && \text{(def.)}
\end{aligned}$$

For the reverse inclusion, we prove by induction on $n$ that for all $h$ and $h'$ satisfying $h' \in (\llbracket \overline{A} \rrbracket_H \ominus G^n)(h)$, there exists $x \in (\llbracket A \rrbracket \diamond \llbracket g \rrbracket)^{\underline{n}} \diamond \llbracket \overline{A} \rrbracket$ for which $h' \in \llbracket x \rrbracket_H(h)$. When $n = 0$ then $h' = h$ and there is some $\alpha \in \overline{A}$ for which $\llbracket \alpha \rrbracket_H(h) = h$. Take $x = \alpha \cdot \pi_\alpha \in \llbracket \overline{A} \rrbracket$ to conclude. When $n = n' + 1$, let $h'' \in G(h)$ witness $h' \in (\llbracket \overline{A} \rrbracket \ominus G^{n'})(h'')$. Then by the IH there exists $x \in (\llbracket A \rrbracket \diamond \llbracket g \rrbracket)^{\underline{n}} \diamond \llbracket \overline{A} \rrbracket$ such that $h' \in \llbracket x \rrbracket_H(h'')$. Furthermore by the IH on $g$, $G(h) \triangleq \llbracket A \cdot g \rrbracket_H(h) \subseteq \llbracket g \rrbracket_H(h) = \bigcup_{z \in \llbracket g \rrbracket} \llbracket z \rrbracket_H(h)$, so since $h'' \in G(h)$ let $w \in \llbracket A \rrbracket \diamond \llbracket g \rrbracket$ witness $h'' \in \llbracket w \rrbracket_H(h)$ (we strengthen $w \in \llbracket g \rrbracket$ to the hypothesis $w \in \llbracket A \rrbracket \diamond \llbracket g \rrbracket$ since otherwise $w \in \llbracket \overline{A} \rrbracket \diamond \llbracket g \rrbracket$ and $h'' \in \llbracket A \cdot g \rrbracket_H(h)$ would not be in $\llbracket w \rrbracket_H(h)$). Then $w \diamond x \in (\llbracket A \rrbracket \diamond \llbracket g \rrbracket)^{\underline{n'+1}} \diamond \llbracket \overline{A} \rrbracket$ satisfies $h' \in \llbracket w \diamond x \rrbracket_H(h)$. Therefore the reverse inclusion holds and we are done. □

*Proof of Lemma 2.4.* We need to show that if $x, y \in \mathrm{GT}$ then $\llbracket x \rrbracket_H = \llbracket y \rrbracket_H$ if and only if $x = y$. The backwards direction is trivial, so suppose $\llbracket x \rrbracket_H = \llbracket y \rrbracket_H$. We show by induction on $x$ that $x = y$.

If $x = \alpha \cdot \pi_\beta \in \mathrm{GT}$, then notice immediately that $y$ cannot contain **dup**, i.e. $y \in \mathrm{At} \cdot \Pi$. Otherwise, if $y = \delta \cdot \pi_\gamma \cdot \mathbf{dup} \cdot w$, then for $pk$ satisfying $\delta \sim pk$ and $pk'$ satisfying $\gamma \sim pk'$, we know $[\![y]\!]_H(pk :: l) = [\![w]\!]_H(pk' :: (pk' :: l))$ whose sole element (which we know exists since $w \in \Pi \cdot (\mathbf{dup} \cdot \Pi)^*$ cannot drop its argument) has length at least one longer than the length of the sole element of $[\![x]\!]_H(pk :: l)$ (if the latter is nonempty at all) which witnesses a contradiction to the assumption $[\![x]\!]_H = [\![y]\!]_H$. Therefore $y = \delta \cdot \pi_\gamma$ for some $\delta, \gamma$. Suppose $\alpha \neq \delta$ and still let $\delta \sim pk$, $\gamma \sim pk'$. Then

$$[\![y]\!]_H(pk :: l) = [\![\delta \cdot \pi_\gamma]\!]_H(pk :: l) = \{pk' :: l\} \neq \emptyset = [\![\alpha \cdot \pi_\beta]\!]_H(h) = [\![x]\!]_H(h)$$

contradicting that $[\![x]\!]_H = [\![y]\!]_H$. Finally suppose $y = \alpha \cdot \pi_\gamma$ for $\gamma \neq \beta$, let $\alpha \sim pk$ and $\gamma \sim pk'$ and $\beta \sim pk''$ so that $pk' \neq pk''$.

$$[\![y]\!]_H(pk :: l) = [\![\alpha \cdot \pi_\gamma]\!]_H(pk :: l) = \{pk' :: l\} \neq \{pk'' :: l\} = [\![\alpha \cdot \pi_\beta]\!]_H(h) = [\![x]\!]_H(h)$$

contradicting that $[\![x]\!]_H = [\![y]\!]_H$.

In the inductive case, let $x = \alpha \cdot \pi_\beta \cdot \mathbf{dup} \cdot w$. Note by similar reasoning as before that $y$ must contain **dup**, so write $y = \delta \cdot \pi_\gamma \cdot \mathbf{dup} \cdot z$. Notice that $\alpha = \delta$ holds by similar reasoning as before: if not then $x$ drops any packet satisfying $\delta$ and vice versa, contradicting $[\![x]\!]_H = [\![y]\!]_H$. Furthermore note that $\pi_\gamma = \pi_\beta$ since, if not then, with $\alpha \sim pk$ and the packets $pk', pk''$ corresponding to $\gamma, \beta$ respectively, $pk'$ and $pk''$ are distinct and would differ in the sole packet history, i.e. $[\![\alpha \cdot \pi_\beta \cdot \mathbf{dup} \cdot w]\!]_H(pk :: l) \neq [\![\alpha \cdot \pi_\gamma \cdot \mathbf{dup} \cdot z]\!]_H(pk :: l)$ a contradiction. Therefore $x = \alpha \cdot \pi_\beta \cdot \mathbf{dup} \cdot w$ and $y = \alpha \cdot \pi_\beta \cdot \mathbf{dup} \cdot z$, so to prove $x = y$ it suffices to prove $w = z$. Note that $[\![w]\!]_H = [\![z]\!]_H$ is also easily provable, since $w, z \in \Pi \cdot (\mathbf{dup} \cdot \Pi)^*$ each

satisfy $[\![z]\!]_H(pk :: l) = \{pk_m :: \ldots :: pk_1 :: l\}$ and $[\![w]\!]_H(pk :: l) = \{pk'_r :: \ldots :: pk'_1 :: l\}$, so for these functions to be equal pointwise it must be that $r = m$ and $pk_i = pk'_i$ for $i \in \{1, 2, \ldots, m\}$. Therefore $[\![w]\!]_H = [\![z]\!]_H$, so the IH yields $w = z$. Therefore $x = y$ as desired. □

*Proof of Lemma 2.5.* We need to prove $[\![x]\!]_H = [\![y]\!]_H$ if and only if $[\![x]\!] = [\![y]\!]$ for all $x, y \in \mathbf{GExp}^-$.

For the backwards direction, Lemma 2.5 says

$$[\![x]\!]_H(h) = \bigcup_{z \in [\![x]\!]} [\![z]\!]_H(h) = \bigcup_{z \in [\![y]\!]} [\![z]\!]_H(h) = [\![y]\!]_H(h)$$

for all $h$ so $[\![x]\!]_H = [\![y]\!]_H$. For the forwards direction, let $[\![x]\!]_H = [\![y]\!]_H$. Suppose $[\![x]\!] \neq [\![y]\!]$ and without loss of generality let $t \in [\![x]\!] \setminus [\![y]\!]$. Let $\alpha$ be the leading complete assignment in $t$. Notice that $\alpha$ does not prefix a trace in $y$: if it prefixed $t' \in [\![y]\!]$ then $t \neq t'$, so for $\alpha \sim pk$ we would have

$$[\![x]\!]_H(pk :: l) = \bigcup_{z \in [\![x]\!]} [\![z]\!]_H(pk :: l) = [\![t]\!]_H(pk :: l) \neq [\![t']\!]_H(pk :: l)$$

$$= \bigcup_{z \in [\![y]\!]} [\![z]\!]_H(pk :: l) = [\![y]\!]_H(pk :: l)$$

which is a contradiction to $[\![x]\!]_H = [\![y]\!]_H$. The unequality $[\![t]\!]_H(pk :: l) \neq [\![t']\!]_H(pk :: l)$ above follows since $t \neq t' \in \mathrm{GT}$ both begin with $\alpha$ and $\alpha \sim pk$, so $t$ and $t'$ differ in either the number of **dup**'s or a complete assignment or both, either of which would witness the unequality between the sole elements of $[\![t]\!]_H(pk :: l)$ and $[\![t']\!]_H(pk :: l)$. The second and penultimate equalities above are determinacy of

$[\![x]\!], [\![y]\!]$ respectively. Therefore $\alpha$ prefixes $t \in [\![x]\!] \setminus [\![y]\!]$ and $\alpha$ does not prefix any $t' \in [\![y]\!]$. Hence

$$[\![x]\!](pk :: l) = \bigcup_{z \in [\![x]\!]} [\![z]\!]_H(h) = [\![t]\!]_H(h) \neq \emptyset$$

$$= \bigcup_{z \in [\![y]\!]} [\![z]\!]_H(h) = [\![y]\!]_H(h)$$

is a contradiction to $[\![x]\!]_H = [\![y]\!]_H$. We are done. □

## A.2 Chapter 3

*Proof of Lemma 3.1.*

(a) That $L \diamond (H \cup K) = (L \diamond H) \cup (L \diamond K)$ and $(L \cup H) \diamond K = (L \diamond K) \cup (H \diamond K)$ both hold are simple subset containment arguments.

(b) That $\diamond$ is associative on languages follows from its associativity on guarded strings. Assume that the following string is defined for $\alpha, \beta, \gamma \in$ At and $\pi, \tau, \sigma \in \Pi$:

$$s \triangleq \alpha \cdot x \cdot \pi \diamond (\beta \cdot y \cdot \tau \diamond \gamma \cdot z \cdot \sigma)$$

For this to be defined, $\beta \cdot y \cdot \tau \diamond \gamma \cdot z \cdot \sigma$ must be defined as well. Note that the string $\beta \cdot y \cdot \tau \diamond \gamma \cdot z \cdot \sigma$ must begin with $\beta$, so for $s$ to be defined it must be that $\alpha_\pi = \beta$, in other words that $\alpha \cdot x \cdot \pi \diamond \beta \cdot \gamma \cdot \tau$ must also be defined

and end in $\tau$. Furthermore, since $\beta \cdot y \cdot \tau \diamond \gamma \cdot z \cdot \sigma$ is defined, we must have $\alpha_\tau = \gamma$, so $(\alpha \cdot x \cdot \pi \diamond \beta \cdot y \cdot \tau) \diamond \gamma \cdot z \cdot \sigma$ must be defined. From this data, if $s$ is defined then $s = \alpha \cdot x \cdot y \cdot z \cdot \sigma$ (concatenation is associative). Computing $(\alpha \cdot x \cdot \pi \diamond \beta \cdot y \cdot \tau) \diamond \gamma \cdot z \cdot \sigma$ yields $\alpha \cdot x \cdot y \cdot z \cdot \sigma = s$ as well, so associativity holds. Then, that $L \diamond (H \diamond K) = (L \diamond H) \diamond K$ holds follows from subset inclusion in both directions with associativity of $\diamond$ on guarded traces.

(c) Let $I = \{\alpha \cdot \pi_\alpha : \alpha \in \mathrm{At}\}$ and $K \subseteq \mathrm{GT}$. To see $I \diamond K = K$, let $\alpha \in \mathrm{At}$. For each $x \in (\Pi \cdot \mathbf{dup})^*, \pi \in \Pi$ for which $\alpha \cdot x \cdot \pi \in K$, it holds that $\alpha \cdot \pi_\alpha \diamond \alpha \cdot x \cdot \pi = \alpha \cdot x \cdot \pi$ so $\alpha \cdot x \cdot \pi \in I \diamond K$. Therefore $K \subseteq I \diamond K$. For the reverse inclusion, let $\alpha \cdot \pi_\alpha \diamond \beta \cdot x \cdot \pi \in I \diamond K$. Then it holds that $\alpha = \beta$ and $\alpha \cdot \pi_\alpha \diamond \alpha \cdot x \cdot \pi = \alpha \cdot x \cdot \pi \in K$. Therefore the reverse inclusion holds, so $I \diamond K = K$. The equality $K \diamond I = K$ is argued in a similar fashion. □

*Proof of Theorem 3.3 (Soundness).*

(U1.) We need to show that $[\![e +_b e]\!] = [\![e]\!]$. Find

$$[\![e +_b e]\!] = [\![b]\!] \diamond [\![e]\!] \cup [\![\overline{b}]\!] \diamond [\![e]\!] = ([\![b]\!] \cup [\![\overline{b}]\!]) \diamond [\![e]\!] = I \diamond [\![e]\!] = [\![e]\!]$$

(U2.) We need to show that $[\![e +_b f]\!] = [\![f +_{\overline{b}} e]\!]$. Find

$$[\![e +_b f]\!] = [\![b]\!] \diamond [\![e]\!] \cup [\![\overline{b}]\!] \diamond [\![f]\!] = [\![\overline{b}]\!] \diamond [\![f]\!] \cup [\![\overline{\overline{b}}]\!] \diamond [\![e]\!] = [\![f +_{\overline{b}} e]\!]$$

since union is commutative and $\overline{\overline{b}} \equiv_{\mathbf{BExp}} b$ and $[\![\cdot]\!]$ is sound on **BExp** with respect to the Boolean axioms (Remark 3.2).

(U3.) We need to show that $[\![(e +_b f) +_c g]\!] = [\![e +_{bc} (f +_c g)]\!]$.

$$\begin{aligned}
[\![(e +_b f) +_c g]\!] &= [\![c]\!] \diamond [\![e +_b f]\!] \cup [\![\overline{c}]\!] \diamond [\![g]\!] \\
&= [\![c]\!] \diamond ([\![b]\!] \diamond [\![e]\!] \cup [\![\overline{b}]\!] \diamond [\![f]\!]) \cup [\![\overline{c}]\!] \diamond [\![g]\!] \\
&= [\![b]\!] \diamond [\![c]\!] \diamond [\![e]\!] \cup [\![\overline{b}]\!] \diamond [\![c]\!] \diamond [\![f]\!] \cup [\![\overline{c}]\!] \diamond [\![g]\!]
\end{aligned}$$

since $\diamond$ is associative and $[\![b]\!] \diamond [\![c]\!] = [\![c]\!] \diamond [\![b]\!]$ for all $b, c \in \mathbf{BExp}$. Next, note that $\overline{b}c \equiv_{\mathbf{BExp}} \overline{bc} \cdot c$ so $[\![\overline{b}c]\!] = [\![\overline{bc} \cdot c]\!]$ by soundness on $\mathbf{BExp}$. Furthermore $\overline{c} \leq \overline{bc}$, so $\overline{c} \vee \overline{bc} \equiv_{\mathbf{BExp}} \overline{bc} \iff [\![\overline{c}]\!] \cup [\![\overline{bc}]\!] = [\![\overline{bc}]\!] \iff [\![\overline{c}]\!] \subseteq [\![\overline{bc}]\!]$. Therefore $[\![\overline{c}]\!] \diamond [\![\overline{bc}]\!] = [\![\overline{c}]\!]$. Proceeding from the last line above,

$$\begin{aligned}
&= [\![bc]\!] \diamond [\![e]\!] \cup [\![\overline{b}c]\!] \diamond [\![f]\!] \cup [\![\overline{c}]\!] \diamond [\![g]\!] \\
&= [\![bc]\!] \diamond [\![e]\!] \cup [\![\overline{bc}]\!] \diamond [\![c]\!] \diamond [\![f]\!] \cup [\![\overline{bc}]\!] \diamond [\![\overline{c}]\!] \diamond [\![g]\!] \\
&= [\![bc]\!] \diamond [\![e]\!] \cup [\![\overline{bc}]\!] \diamond ([\![c]\!] \diamond [\![f]\!] \cup [\![\overline{c}]\!] \diamond [\![g]\!]) \\
&= [\![bc]\!] \diamond [\![e]\!] \cup [\![\overline{bc}]\!] \diamond [\![f +_c g]\!] \\
&= [\![e +_{bc} (f +_c g)]\!]
\end{aligned}$$

as needed.

(U4.) We need to show $[\![e +_b g]\!] = [\![be +_b g]\!]$. Find

$$[\![e +_b g]\!] = [\![b]\!] \diamond [\![e]\!] \cup [\![\overline{b}]\!] \diamond [\![g]\!] = ([\![b]\!] \diamond [\![b]\!]) \diamond [\![e]\!] \cup [\![\overline{b}]\!] \diamond [\![g]\!] = [\![be +_b g]\!]$$

since $[\![b]\!] = [\![b]\!] \diamond [\![b]\!]$ and $\diamond$ is associative.

(U5.) We need to show that $[\![eg +_b fg]\!] = [\![(e +_b f)g]\!]$. Find

$$
\begin{aligned}
[\![eg +_b fg]\!] &\equiv [\![b]\!] \diamond [\![eg]\!] \cup [\![\overline{b}]\!] \diamond [\![fg]\!] \\
&= [\![b]\!] \diamond [\![e]\!] \diamond [\![g]\!] \cup [\![\overline{b}]\!] \diamond [\![f]\!] \diamond [\![g]\!] \\
&= ([\![b]\!] \diamond [\![e]\!] \cup [\![\overline{b}]\!] \diamond [\![f]\!]) \diamond [\![g]\!] \\
&= [\![e +_b f]\!] \diamond [\![g]\!] \\
&= [\![(e +_b f)g]\!]
\end{aligned}
$$

as needed.

(S1.) We need to show $[\![(ef)g]\!] = [\![e(fg)]\!]$. This follows immediately from associativity of $\diamond$.

(S2.) We need to show $[\![0 \cdot e]\!] = [\![0]\!]$. Note that $[\![0]\!] = \emptyset$ since no $\alpha \in \mathrm{At}$ satisfies $\alpha \leq 0$, hence

$$[\![0 \cdot e]\!] = [\![0]\!] \cdot [\![e]\!] = \emptyset \diamond [\![e]\!] = \emptyset = [\![0]\!]$$

(S3.) Identical to proof of (S2.).

(S4.) We need to show $[\![1 \cdot e]\!] = [\![e]\!]$. This is immediate since $[\![1]\!] = I$, the identity of the monoid $(2^{\mathrm{GT}}, \diamond)$ as shown in Lemma 3.1, so $[\![1 \cdot e]\!] = I \diamond [\![e]\!] = [\![e]\!]$.

(S5.) Identical to proof of (S4.).

(W1.) We need to show $[\![ee^{(b)} +_b 1]\!] = [\![e^{(b)}]\!]$.

$$
\begin{aligned}
[\![ee^{(b)} +_b 1]\!] &= [\![b]\!] \diamond [\![ee^{(b)}]\!] \cup [\![\overline{b}]\!] \diamond [\![1]\!] \\
&= [\![b]\!] \diamond [\![e]\!] \diamond [\![e^{(b)}]\!] \cup [\![\overline{b}]\!] \diamond I \\
&= [\![b]\!] \diamond [\![e]\!] \diamond (\bigcup_{n\in\mathbb{N}} ([\![b]\!] \diamond [\![e]\!])^{\underline{n}}) \diamond [\![\overline{b}]\!] \cup [\![\overline{b}]\!] \\
&= \left(\bigcup_{n\in\mathbb{N}} ([\![b]\!] \diamond [\![e]\!])^{\underline{n+1}}\right) \diamond [\![\overline{b}]\!] \cup [\![\overline{b}]\!] \\
&= \left(\left(\bigcup_{n\in\mathbb{N}} ([\![b]\!] \diamond [\![e]\!])^{\underline{n+1}}\right) \cup I\right) \diamond [\![\overline{b}]\!] \\
&= (\bigcup_{n\in\mathbb{N}} ([\![b]\!] \diamond [\![e]\!])^{\underline{n}}) \diamond [\![\overline{b}]\!] \\
&= [\![e^{(b)}]\!]
\end{aligned}
$$

where we recall that $L^{\underline{0}} \triangleq I$ for any $L \in 2^{\mathrm{GT}}$.

(W2.) This proof follows the appendix of [11]. We need to prove $[\![(ce)^{(b)}]\!] = [\![(e +_c 1)^{(b)}]\!]$. First, we will show that for all $n \geq 0$, if $x \in ([\![b]\!] \diamond [\![c]\!] \diamond [\![e]\!] \cup [\![\overline{c}]\!])^{\underline{n}}$, then there exists some $m \leq n$ for which $x \in ([\![b]\!] \diamond [\![c]\!] \diamond [\![e]\!])^{\underline{m}}$, by induction on $n$. For the case $n = 0$, we know $x \in I$, for which we set $m \triangleq 0$ to finish the claim. For $n > 0$, write

$$
x = y \diamond z, \quad y \in [\![b]\!] \diamond [\![c]\!] \diamond [\![e]\!] \cup [\![\overline{c}]\!], \quad z \in ([\![b]\!] \diamond [\![c]\!] \diamond [\![e]\!] \cup [\![\overline{c}]\!])^{\underline{n-1}}
$$

By the IH, $z \in ([\![b]\!] \diamond [\![c]\!] \diamond [\![e]\!])^{\underline{k}}$ for some $k \leq n - 1$. If $y \in [\![\overline{c}]\!] \subseteq I$, then $x = y \diamond z = z \in ([\![b]\!] \diamond [\![c]\!] \diamond [\![e]\!])^{\underline{k}}$ and we are done with $m \triangleq k$. Otherwise, if $y \in [\![b]\!] \diamond [\![c]\!] \diamond [\![e]\!]$ holds then

$$
x = y \diamond z \in [\![b]\!] \diamond [\![c]\!] \diamond [\![e]\!] \diamond ([\![b]\!] \diamond [\![c]\!] \diamond [\![e]\!])^{\underline{k}} = ([\![b]\!] \diamond [\![c]\!] \diamond [\![e]\!])^{\underline{k+1}}
$$

so we set $m \triangleq k + 1 \leq n$ to finish the proof. Using this, find

$$[\![(ce)^{(b)}]\!] \triangleq \bigcup_{n\in\mathbb{N}} ([\![b]\!] \diamond [\![c]\!] \diamond [\![e]\!])^{\underline{n}} \diamond [\![\overline{b}]\!] = \bigcup_{n\in\mathbb{N}} ([\![b]\!] \diamond [\![c]\!] \diamond [\![e]\!] \cup [\![\overline{c}]\!])^{\underline{n}} \diamond [\![\overline{b}]\!] \triangleq R$$

where the latter equality is our derivation above: the $\subseteq$ containment is obvious, and the $\supseteq$ containment is what our claim proved. Now, observe that

$$[\![(e +_c 1)^{(b)}]\!] = \bigcup_{n\in\mathbb{N}} ([\![b]\!] \diamond [\![e +_c 1]\!])^{\underline{n}} \diamond [\![\overline{b}]\!] = \bigcup_{n\in\mathbb{N}} ([\![b]\!] \diamond [\![c]\!] \diamond [\![e]\!] \cup [\![b]\!] \diamond [\![\overline{c}]\!])^{\underline{n}} \diamond [\![\overline{b}]\!] \triangleq S$$

Since $\diamond$ restricted to $\mathcal{P}(I) = \mathcal{P}(\{\alpha \cdot \pi_\alpha : \alpha \in \mathrm{At}\})$ is just intersection, we know $[\![b]\!] \diamond [\![\overline{c}]\!] \subseteq [\![\overline{c}]\!]$ and the containment $S \subseteq R$ follows. Conversely, let $w \in ([\![b]\!] \diamond [\![c]\!] \diamond [\![e]\!] \cup [\![\overline{c}]\!])^{\underline{n}} \diamond [\![\overline{b}]\!]$ for some $n$. Write $w = w_1 \diamond \ldots \diamond w_n \diamond w'$ where $w_i \in [\![b]\!] \diamond [\![c]\!] \diamond [\![e]\!] \cup [\![\overline{c}]\!]$ for each $1 \leq i \leq n$, $w' \in [\![\overline{b}]\!]$. We need to argue $w = y_1 \diamond \ldots \diamond y_m \diamond y'$ for $y_i \in [\![b]\!] \diamond [\![c]\!] \diamond [\![e]\!] \cup [\![b]\!] \diamond [\![\overline{c}]\!], y' \in [\![\overline{b}]\!]$ and some $m$. If $n = 0$ then we are trivially done with $m \triangleq 0$ since $w = w' \in ([\![b]\!] \diamond [\![c]\!] \diamond [\![e]\!] \cup [\![b]\!] \diamond [\![\overline{c}]\!])^{\underline{0}} \diamond [\![b]\!]$, so suppose that $n \geq 1$. We split into cases. If there is no $i \in \{1, \ldots, n\}$ for which $w_i \in [\![\overline{c}]\!]$ then we are done with $m \triangleq n$ because each $w_j$ came from $[\![b]\!] \diamond [\![c]\!] \diamond [\![e]\!]$: the factorization $w = w_1 \diamond \ldots \diamond w_n \diamond w'$ witnesses $w \in ([\![b]\!] \diamond [\![c]\!] \diamond [\![e]\!] \cup [\![b]\!] \diamond [\![\overline{c}]\!])^{\underline{n}} \diamond [\![\overline{b}]\!] \subseteq S$.

Otherwise there exists a minimal $i \in \{1, \ldots, n\}$ for which $w_i \in [\![\overline{c}]\!]$. Therefore $w_j \in [\![\overline{c}]\!]$ for all $j \geq i$. Thus

$$w = w_1 \diamond \ldots \diamond w_n \diamond w' = w_1 \diamond \ldots \diamond w_i \diamond w'$$

where $j < i$ implies $w_j \in [\![b]\!] \diamond [\![c]\!] \diamond [\![e]\!]$. We now case analysis on whether $w_i \in [\![b]\!]$. If yes then we have a contradiction: $w' \in [\![\overline{b}]\!]$ so the subterm

$w_i \diamond w'$ would be undefined. Therefore we are forced to yield $w_i \in [\![\overline{b}]\!]$ so $w_i \diamond w' = w_i$. Therefore

$$w = w_1 \diamond \ldots \diamond w_i \diamond w' = w_1 \diamond \ldots \diamond w_i$$

We have that, for all $1 \leq j \leq i-1$, it holds that $w_j \in [\![b]\!] \diamond [\![c]\!] \diamond [\![e]\!]$ and $w_i \in [\![\overline{b}]\!]$. Therefore $m \triangleq i - 1 \geq 0$ witnesses $w \in ([\![b]\!] \diamond [\![c]\!] \diamond [\![e]\!] \cup [\![b]\!] \diamond [\![\overline{c}]\!])^{\underline{m}} \diamond [\![\overline{b}]\!]$. This shows the containment $R \subseteq S$, so $R = S$ and we are done. □

*Proof of Definition 3.4 (well-definedness of atom-indexed sum).* The following proof is given in Part B of the Appendix in [11]. Given $\Phi \subseteq \mathrm{At}$, the operator $\{e_\alpha\}_{\alpha\in\Phi} \mapsto +_{\alpha\in\Phi} e_\alpha$ is well-defined up to $\equiv$. We proceed by induction on the size of $\Phi$. If $\Phi = \emptyset$ or $\Phi = \{\alpha\}$ then the proof is immediate, since the respective expressions must be equal to 0 and $e_\alpha +_\alpha 0$ (not $e_\alpha$ as [11] claims). For the inductive step, we show that $\beta, \gamma \in \Phi$ implies $e_\beta +_\beta \left(+_{\alpha\in\Phi\setminus\{\beta\}} e_\beta\right) \equiv e_\gamma +_\gamma \left(+_{\alpha\in\Phi\setminus\{\gamma\}} e_\alpha\right)$. Find

$$\begin{aligned}
e_\beta +_\beta \left(+_{\alpha\in\Phi\setminus\{\beta\}} e_\alpha\right) &\equiv e_\beta +_\beta \left(e_\gamma +_\gamma \left(+_{\alpha\in\Phi\setminus\{\beta,\gamma\}} e_\alpha\right)\right) && \text{(induction)}\\
&\equiv (e_\beta +_\beta e_\gamma) +_{\beta+\gamma} \left(+_{\alpha\in\Phi\setminus\{\beta,\gamma\}} e_\alpha\right) && \text{(U3′.)}\\
&\equiv (e_\gamma +_{\overline{\beta}} e_\beta) +_{\beta+\gamma} \left(+_{\alpha\in\Phi\setminus\{\beta,\gamma\}} e_\alpha\right) && \text{(U2.)}\\
&\equiv e_\gamma +_{\overline{\beta}(\beta+\gamma)} \left(e_\beta +_{\beta+\gamma} \left(+_{\alpha\in\Phi\setminus\{\beta,\gamma\}} e_\alpha\right)\right) && \text{(U3.)}\\
&\equiv e_\gamma +_\gamma \left(e_\beta +_\gamma \left(e_\beta +_\beta \left(+_{\alpha\in\Phi\setminus\{\beta,\gamma\}} e_\alpha\right)\right)\right) && \text{(BA), (U3′.), (U1.)}\\
&\equiv e_\gamma +_\gamma \overline{\gamma}\left(e_\beta +_\gamma \left(e_\beta +_\beta \left(+_{\alpha\in\Phi\setminus\{\beta,\gamma\}} e_\alpha\right)\right)\right) && \text{(U4′.)}\\
&\equiv e_\gamma +_\gamma \left(e_\beta +_\beta \left(+_{\alpha\in\Phi\setminus\{\beta,\gamma\}} e_\alpha\right)\right) && \text{(U2.), (U8.)}\\
&\equiv e_\gamma +_\gamma \left(+_{\alpha\in\Phi\setminus\{\gamma\}} e_\alpha\right) && \text{(induction)}
\end{aligned}$$

where BA stands for Boolean algebra and we combine (U3′.) and (U1.) to show the (universally quantified) fact $e +_{\alpha+\beta} g \equiv e +_\alpha (e +_\beta g)$. Specifically, (U1.)

shows $e \equiv e +_\alpha e$, therefore $e +_{\alpha+\beta} g \equiv (e +_\alpha e) +_{\alpha+\beta} g$, from which applying (U3′.) with $e \triangleq e, f \triangleq e, g \triangleq g$ shows $e +_{\alpha+\beta} g \equiv (e +_\alpha e) +_{\alpha+\beta} g \equiv e +_\alpha (e +_\beta g)$. □

## A.3 Chapter 4

*Proof of Lemma 4.5.* We need to show that if $x \sim y$ are bisimilar states in $H$-coalgebras $\mathcal{X} = (X, \delta_\mathcal{X})$ and $\mathcal{Y} = (Y, \delta_\mathcal{Y})$ then $l^\mathcal{X}(x) = l^\mathcal{Y}(y)$. We proceed to show for all $t \in \mathrm{GT}$ that $t \in l^\mathcal{X}(x) \iff t \in l^\mathcal{Y}(y)$ by induction on $t$. Recall that bisimilarity is an equivalence relation [7].

When $t = \alpha \cdot \pi_\beta$,

$$\begin{aligned} \mathrm{accept}(x, \alpha \cdot \pi_\beta) &\iff \delta_\mathcal{X}(x)(\alpha) = \beta && \text{(def.)} \\ &\iff \delta_\mathcal{Y}(y)(\alpha) = \beta && \text{(hyp.)} \\ &\iff \mathrm{accept}(y, \alpha \cdot \pi_\beta) && \text{(def.)} \end{aligned}$$

so $t \in l^\mathcal{X}(x) \iff t \in l^\mathcal{Y}(y)$. We are done in the base case.

When $t = \alpha \cdot \pi_\beta \cdot \mathbf{dup} \cdot w$,

$$\mathrm{accept}(x, \alpha \cdot \pi_\beta \cdot \mathbf{dup} \cdot w) \iff \exists x'.\, \delta_\mathcal{X}(x)(\alpha) = (\beta, x') \wedge \mathrm{accept}(x', \beta \cdot w)$$

If there exists $x' \in X$ for which $\delta_\mathcal{X}(x)(\alpha) = (\beta, x')$ then, since $x \sim y$, we know there exists $y' \in Y$ such that $x' \sim y'$ and $\delta_\mathcal{Y}(y)(\alpha) = (\beta, y')$. Since $x' \sim y'$, by the IH on $w$ we know that $\mathrm{accept}(x', \beta \cdot w) \iff \mathrm{accept}(y', \beta \cdot w)$. The analogous argument holds in reverse as well, since bisimilarity is symmetric, so

$$\exists x'.\, \delta_X(x)(\alpha) = (\beta, x') \wedge \mathrm{accept}(x', \beta \cdot w) \iff \exists y'.\, \delta_Y(y)(\alpha) = (\beta, y') \wedge \mathrm{accept}(y', \beta \cdot w)$$
$$\iff \mathrm{accept}(y, \alpha \cdot \pi_\beta \cdot \mathbf{dup} \cdot w)$$

Therefore $t \in l^X(x) \iff t \in l^Y(y)$. We are done with the inductive step. □

*Proof of Theorem 4.7.* We follow the proof structure for Theorem 5.8 in [11]. We need to verify three conditions: (i) that $l^X(x) \in \mathcal{L}$ for every $x \in X$; (ii) that $l^X : X \to \mathcal{L}$ is a homomorphism; and (iii) that $l^X$ is the unique such homomorphism.

(i) We need to show that for all $x \in X$ and for all $\alpha \in \mathrm{At}$ that there exists at most one string prefixed with $\alpha$ in $l^X(x)$. We show that, for all $x \in X$, if $w, w' \in l^X(x)$ both start with $\alpha$ then $w = w'$. We proceed by induction on $w \in \mathrm{GT}$. Note that, by definition of the acceptance condition (in particular well-definedness of $\delta_X(x)(\alpha)$), either both $w$ and $w'$ contain **dup** or neither do, since if not then we immediately find a contradiction.

If neither does, let $w = \alpha \cdot \pi$ and $w' = \alpha \cdot \pi'$. Then $\mathrm{accept}(x, w) \iff \delta_X(x)(\alpha) = \alpha_\pi$ and likewise $\mathrm{accept}(x, w') \iff \delta_X(x)(\alpha) = \alpha_{\pi'}$. Therefore $\alpha_\pi = \alpha_{\pi'}$, so $\pi = \pi'$ and $w = w'$.

For the inductive step, i.e. for the case where $w = \alpha \cdot \pi \cdot \mathbf{dup} \cdot p$ and $w' = \alpha \cdot \tau \cdot \mathbf{dup} \cdot q$ for $p, q \in \Pi \cdot (\mathbf{dup} \cdot \Pi)^*$, note that

$$\mathrm{accept}(x, w) \iff \exists y.\, \delta_X(x)(\alpha) = (\beta, y) \wedge \mathrm{accept}(y, \beta \cdot p)$$

from which we necessarily infer $\beta = \alpha_\pi$. Note that if the pair $(\beta, y)$ satisfying $\delta_X(x)(\alpha) = (\beta, y)$ exists then it is unique by well-definedness of $\delta_X$. Likewise

$$\mathrm{accept}(x, w') \iff \exists y.\, \delta_X(x)(\alpha) = (\beta, y) \wedge \mathrm{accept}(y, \beta \cdot q)$$

from which we necessarily infer that $\beta = \alpha_\tau$ so (by well-definedness of $\delta_X(x)(\alpha)$) we know $\pi = \tau$. Therefore it remains to show that $p = q$. We apply the IH on $\beta \cdot p, \beta \cdot q \in l^Y(y)$ to learn $\beta \cdot p = \beta \cdot q$, so $p = q$ and we are done.

(ii) We now need to show that $l^X : X \to \mathcal{L}$ is a homomorphism. Let $x \in X, \alpha \in \mathrm{At}$. We first verify that $\delta_X(x)(\alpha) \in 1 + \mathrm{At}$ implies $\delta_{\mathcal{L}}(l^X(x))(\alpha) = \delta_X(x)(\alpha)$. If $\delta_X(x)(\alpha) = \bot$ then, in particular $x$ cannot accept any string prefixed with $\alpha$ (by definition of the acceptance condition). Therefore there is no $y$ for which $\alpha \cdot y \in l^X(x)$, which implies $\delta_{\mathcal{L}}(l^X(x))(\alpha) = \bot = \delta_X(x)(\alpha)$. In the case where $\delta_X(x)(\alpha) = \beta$, then necessarily accept$(x, \alpha \cdot \pi_\beta)$ so $\alpha \cdot \pi_\beta \in l^X(x)$ is the unique string prefixed by $\alpha \in l^X(x)$. Therefore, by definition it holds that $\delta_{\mathcal{L}}(l^X(x))(\alpha) = \beta = \delta_X(x)(\alpha)$.

For the case where $\delta_X(x)(\alpha) = (\beta, y)$ then we need to show that $\delta_{\mathcal{L}}(l^X(x))(\alpha) = (\beta, l^X(y))$. For $L \in \mathcal{L}$ let

$$L_{\alpha\beta} = \begin{cases} \emptyset & \text{if } \nexists y.\, \alpha \cdot \pi_\beta \cdot \mathbf{dup} \cdot y \in L \\ \{\beta \cdot x\} & \text{if } \alpha \cdot \pi_\beta \cdot \mathbf{dup} \cdot x \in L \end{cases}$$

Notice that $|L_{\alpha\beta}| \leq 1$. Before proceeding, we prove the following implication:

$$(*) \quad \delta_X(s)(\alpha) = (\beta, t) \implies l^X(s)_{\alpha\beta} = l^X(t)$$

The claim $(*)$ is seen as follows: given the premise, it holds that

$$\beta \cdot x \in l^X(s)_{\alpha\beta} \iff \alpha \cdot \pi_\beta \cdot \mathbf{dup} \cdot x \in l^X(s)$$

$$\iff \beta \cdot x \in l^{\mathcal{X}}(t)$$

by definition, given that $\delta_{\mathcal{X}}(s)(\alpha) = (\beta, t)$ holds. Furthermore $(X, \delta_{\mathcal{X}})$ is strongly normal: since $\delta_{\mathcal{X}}(s)(\alpha) = (\beta, t)$ then $l^{\mathcal{X}}(t)$ must be a singleton set whose sole element begins with $\beta$. Therefore $l^{\mathcal{X}}(s)_{\alpha\beta} = \{\beta \cdot x\} = l^{\mathcal{X}}(t)$ for some $x$, assuming $\delta_{\mathcal{X}}(s)(\alpha) = (\beta, t)$. We have shown that $(*)$ holds.

We now proceed with the proof that $\delta_{\mathcal{X}}(x)(\alpha) = (\beta, y)$ implies $\delta_{\mathcal{L}}(l^{\mathcal{X}}(x))(\alpha) = (\beta, l^{\mathcal{X}}(y))$. By strong normality of $\mathcal{X}$, let $l^{\mathcal{X}}(y) = \{\beta \cdot w\}$ for some $w \in \Pi \cdot (\mathbf{dup} \cdot \Pi)^*$. Therefore

$$\begin{aligned}
& \alpha \cdot \pi_\beta \cdot \mathbf{dup} \cdot w \in l^{\mathcal{X}}(x) && (\text{def. } l^{\mathcal{X}}(x)) \\
\implies\ & l^{\mathcal{X}}(x)_{\alpha\beta} = \{\beta \cdot w\} && (\text{def. } l^{\mathcal{X}}(x)_{\alpha\beta}) \\
\implies\ & \delta_{\mathcal{L}}(l^{\mathcal{X}}(x))(\alpha) = (\beta, l^{\mathcal{X}}(x)_{\alpha\beta}) && (\text{def. } \delta_{\mathcal{L}}) \\
\implies\ & \delta_{\mathcal{L}}(l^{\mathcal{X}}(x))(\alpha) = (\beta, l^{\mathcal{X}}(y)) && (*)
\end{aligned}$$

Therefore $l^{\mathcal{X}} : X \to \mathcal{L}$ is a homomorphism of $H$-coalgebras.

(iii) We now wish to show that $l^{\mathcal{X}} : X \to \mathcal{L}$ is the unique homomorphism of $H$-coalgebras $(X, \delta_{\mathcal{X}}) \to (\mathcal{L}, \delta_{\mathcal{L}})$. Let $L : X \to \mathcal{L}$ be any homomorphism of $H$-coalgebras. We will show by induction on $w \in \mathrm{GT}$ that $w \in L(x) \iff w \in l^{\mathcal{X}}(x)$.

When $w = \alpha \cdot \pi_\beta$,

$$\begin{aligned}
\alpha \cdot \pi_\beta \in L(x) \iff\ & \delta_{\mathcal{L}}(L(x))(\alpha) = \beta && (\text{def. } \delta_{\mathcal{L}}) \\
\iff\ & \delta_{\mathcal{X}}(x)(\alpha) = \beta && (L \text{ hom.}) \\
\iff\ & \alpha \cdot \pi_\beta \in l^{\mathcal{X}}(x) && (\text{def. } l^{\mathcal{X}}(x))
\end{aligned}$$

When $w = \alpha \cdot \pi_\beta \cdot \mathbf{dup} \cdot w'$,

$$
\begin{aligned}
& \alpha \cdot \pi_\beta \cdot \mathbf{dup} \cdot w' \in L(x) && \\
\iff\ & \delta_{\mathcal{L}}(L(x))(\alpha) = (\beta, \{\beta \cdot w'\}) \wedge L(x)_{\alpha\beta} = \{\beta \cdot w'\} && (\text{def. } \delta_{\mathcal{L}}, L_{\alpha\beta}) \\
\iff\ & \exists y.\, \delta_{\mathcal{X}}(x)(\alpha) = (\beta, y) \wedge L(y) = \{\beta \cdot w'\} && (L \text{ hom.}) \\
\iff\ & \exists y.\, \delta_{\mathcal{X}}(x)(\alpha) = (\beta, y) \wedge l^{\mathcal{X}}(y) = \{\beta \cdot w'\} && (\text{IH, strong normality}) \\
\iff\ & \alpha \cdot \pi_\beta \cdot \mathbf{dup} \cdot w' \in l^{\mathcal{X}}(x) && (\text{def. } l^{\mathcal{X}}(x))
\end{aligned}
$$

Therefore we have shown $w \in L(x) \iff w \in l^{\mathcal{X}}(x)$ for all $w \in \mathrm{GT}$ for all $x \in X$ so $L = l^{\mathcal{X}}$ as morphisms $(X, \delta_{\mathcal{X}}) \to (\mathcal{L}, \delta_{\mathcal{L}})$. □

*Proof of Lemma 4.9.* We proceed by induction on $e$.

Case $e = b$ : Let $\alpha \in \mathrm{At}$. Then $\alpha \cdot \pi_\alpha \in [\![b]\!] \iff \alpha \leq b$. Furthermore $\tilde{D}(b)(\alpha) = \alpha$ if and only if $\alpha \leq b$, so $\alpha \cdot \pi_\alpha \in l^{\tilde{D}}(b)$ if and only if $\alpha \leq b$. Then we are done, since $[\![b]\!] \subseteq \{\alpha \cdot \pi_\alpha : \alpha \in \mathrm{At}\}$ and it is clear from inspection of $D$ that $l^{\tilde{D}}(b) \subseteq \{\alpha \cdot \pi_\alpha : \alpha \in \mathrm{At}\}$. This case holds.

Case $e = f \leftarrow n$ : Then $[\![f \leftarrow n]\!] = \{\alpha \cdot \pi_\alpha[f/n] : \alpha \in \mathrm{At}\}$. Furthermore $\alpha \cdot \pi_\beta \in l^{\tilde{D}}(f \leftarrow n)$ if and only if $\beta = \alpha[f/n]$. It is clear from inspection of $D$ that no string in $l^{\tilde{D}}(f \leftarrow n)$ contains **dup**, so this case holds.

Case $e = \mathbf{dup}$ : Then $[\![\mathbf{dup}]\!] = \{\alpha \cdot \pi_\alpha \cdot \mathbf{dup} \cdot \pi_\alpha : \alpha \in \mathrm{At}\}$. Observe that every string in $l^{\tilde{D}}(\mathbf{dup})$ must contain **dup** by construction. Let $\alpha \in \mathrm{At}$ and suppose $\alpha \cdot \pi_\beta \cdot \mathbf{dup} \cdot w \in l^{\tilde{D}}(\mathbf{dup})$. Then $\tilde{D}(\mathbf{dup})(\alpha) = (\beta, t) \wedge \mathrm{accept}(t, \beta \cdot w)$ for some $t$. But by definition of $D$, $\tilde{D}(\mathbf{dup})(\alpha) = (\alpha, \alpha)$, so $\alpha \cdot \pi_\beta \cdot \mathbf{dup} \cdot w \in l^{\tilde{D}}(\mathbf{dup})$ if and only if $\beta = \alpha$ and $\mathrm{accept}(\alpha, \alpha \cdot w)$. The latter holds if and only if $w = \pi_\alpha$, i.e. $\alpha \cdot \pi_\beta \cdot \mathbf{dup} \cdot w \in l^{\tilde{D}}(\mathbf{dup})$ implies $\beta = \alpha$, $w = \pi_\alpha$. The reverse inclusion is also clear, i.e. one trivially verifies $\mathrm{accept}(\mathbf{dup}, \alpha \cdot \pi_\alpha \cdot \mathbf{dup} \cdot \pi_\alpha)$ holds for all $\alpha \in \mathrm{At}$ by computing $\tilde{D}(\mathbf{dup})(\alpha) = (\alpha, \alpha)$. This case holds.

Case $e = g +_b h$ : Then

$$\beta \cdot w \in [\![g +_b h]\!] \iff \beta \leq b \wedge \beta \cdot w \in [\![g]\!] \vee \beta \leq \overline{b} \wedge \beta \cdot w \in [\![h]\!]$$

Likewise

$$\beta \cdot w \in l^{\tilde{D}}(g +_b h) \iff \text{accept}(g +_b h, \beta \cdot w)$$

We now case analysis on $\beta \cdot w \in \text{GT}$ : we show only the case for $\beta \leq b$, since $\beta \leq \overline{b}$ is similar. If $w$ does not contain **dup**, then let $w = \pi_\gamma$. We have

$$\begin{aligned}
\text{accept}(g +_b h, \beta \cdot \pi_\gamma) &\iff \tilde{D}(g +_b h)(\beta) = \gamma && \text{(def.)} \\
&\iff \tilde{D}(g)(\beta) = \gamma && (\beta \leq b) \\
&\iff \text{accept}(g, \beta \cdot \pi_\gamma) && \text{(def.)} \\
&\iff \beta \cdot \pi_\gamma \in [\![g]\!] && \text{(IH)}
\end{aligned}$$

If $w = \pi_\gamma \cdot \mathbf{dup} \cdot w'$, then

$$\begin{aligned}
& \text{accept}(g +_b h, \beta \cdot \pi_\gamma \cdot \mathbf{dup} \cdot w') \\
\iff\ & \exists s.\, \tilde{D}(g +_b h)(\beta) = (\gamma, s) \wedge \text{accept}(s, \gamma \cdot w') && \text{(def.)} \\
\iff\ & \exists s.\, \tilde{D}(g)(\beta) = (\gamma, s) \wedge \text{accept}(s, \gamma \cdot w') && \beta \leq b \\
\iff\ & \text{accept}(g, \beta \cdot \pi_\gamma \cdot \mathbf{dup} \cdot w') && \text{(def.)} \\
\iff\ & \beta \cdot \pi_\gamma \cdot \mathbf{dup} \cdot w' \in [\![g]\!] && \text{(IH)} \\
\iff\ & \beta \cdot \pi_\gamma \cdot \mathbf{dup} \cdot w' \in [\![g +_b h]\!] && (\beta \leq b)
\end{aligned}$$

which finishes the case $e = g +_b h$ (arguing $\beta \leq \overline{b}$ similarly).

Case $e = g \cdot h$ : Then

$$t \in [\![g \cdot h]\!] \iff \exists y \in [\![g]\!], z \in [\![h]\!]. \, t = y \diamond z$$

By the IHs, $y \in [\![g]\!], z \in [\![h]\!]$ hold if and only if $y \in l^{\tilde{D}}(g)$ and $z \in l^{\tilde{D}}(h)$ respectively. Futhermore $t = y \diamond z$ is defined if and only if the trailing complete assignment in $y$ and the leading atom in $z$ agree. Let $z = \beta \cdot z'$ and $y = y' \cdot \pi_\beta$. Then

$$\begin{aligned}
& \exists y \in [\![g]\!], z \in [\![h]\!]. \, t = y \diamond z \\
\iff \quad & \exists y \in l^{\tilde{D}}(g), z \in l^{\tilde{D}}(h). \, t = y \diamond z && \text{(IH)} \\
\iff \quad & \exists y, z. \, t = y \diamond z \wedge \text{accept}(g, y' \cdot \pi_\beta) \wedge \text{accept}(h, \beta \cdot z') && \text{(def.)} \\
\iff \quad & \exists y, z. \, t = y \diamond z \wedge \text{accept}(g \cdot h, y \diamond z) && (y \diamond z \text{ defined}) \\
\iff \quad & t \in l^{\tilde{D}}(g \cdot h) && \text{(def.)}
\end{aligned}$$

Case $e = g^{(b)}$ : Then

$$t \in [\![g^{(b)}]\!] \iff \exists! n \in \mathbb{N}. \, t \in ([\![b]\!] \diamond [\![g]\!])^{\underline{n}} \diamond [\![\overline{b}]\!]$$

by determinacy. By the IH on $g$, $[\![g]\!] = l^{\tilde{D}}(g)$ holds. Then

$$t \in ([\![b]\!] \diamond [\![g]\!])^{\underline{n}} \diamond [\![\overline{b}]\!] \iff t = y_1 \diamond \ldots \diamond y_n$$

where each $y_i \in [\![b]\!] \diamond [\![g]\!]$ and the leading atom of $y_1$, say $\alpha$, satisfies $\alpha \leq b$ and trailing complete assignment $\pi_{\beta_i}$ in $y_i$ satisfies $\beta_i \leq b$ for $1 \leq i < n$ and $\beta_n \leq \overline{b}$. We show by induction on $t$ that $t \in [\![g^{(b)}]\!] \iff \text{accept}(g^{(b)}, t)$ using the facts above.

Suppose $t = \alpha \cdot \pi_\beta \in [\![g^{(b)}]\!]$. Then $t = y_1 \diamond \ldots \diamond y_n$ according to above. By the IH, $[\![g]\!] = l^{\tilde{D}}(g)$, from which $[\![b \cdot g]\!] = l^{\tilde{D}}(b \cdot g)$ follows by restricting the leading atoms. Therefore letting $y_i = \alpha_i \cdot x_i \cdot \pi_{\beta_i}$ we know $D(b \cdot g)(\alpha_i) = \beta_i = \alpha_{i+1}$ for $1 \leq i < n$ and $D(b \cdot g \cdot \overline{b})(\alpha_n) = \beta_n$ (no transitions between states are taken since no $y_i$ contains **dup**: if one did then so would $t$). It follows by definition now that $\text{accept}(g^{(b)}, y_1 \diamond \ldots \diamond y_n)$.

For the reverse direction $t = \alpha \cdot \pi_\beta \in l^{\tilde{D}}(g^{(b)})$, it is by definition that $\tilde{D}(g^{(b)})(\alpha) = \beta$. If $\alpha \leq \overline{b}$ then $\beta = \alpha$ and $\alpha \cdot \pi_\alpha \in [\![g^{(b)}]\!]$. If $\alpha \leq b$ then $\tilde{D}(g)(\alpha) = \gamma$ and $\tilde{D}(g^{(b)})(\gamma) = \beta$ for some $\gamma$. We defined $D$ as the least upper bound of all functions $\delta : \mathbf{GExp} \to H(\mathbf{GExp})$ in the partial-function extension order (treating outputs of $\bot$ as undefined), so in particular $D$ satisfies an induction principle: we can assume $\gamma \cdot \pi_\beta \in [\![g^{(b)}]\!]$ i.e. the IH on $D$ holds. From this and $\tilde{D}(g)(\alpha) = \gamma$ and the IH on $g$, $\alpha \cdot \pi_\gamma \in [\![g]\!]$ so $\alpha \cdot \pi_\gamma \diamond \gamma \cdot \pi_\beta = \alpha \cdot \pi_\beta \in [\![g^{(b)}]\!]$ (recall $\alpha \leq b$ in this case).

When $t = \alpha \cdot \pi_\beta \cdot \mathbf{dup} \cdot t' \in [\![g^{(b)}]\!]$ let $t = y_1 \diamond \ldots \diamond y_n$ and $k \in \{1, \ldots, n\}$ be the smallest index for which $y_k$ contains **dup**, say $y_k = x \cdot \pi_\gamma \cdot \mathbf{dup} \cdot z$ for $x \in \text{At} \cdot (\Pi \cdot \mathbf{dup})^*, z \in \Pi \cdot (\mathbf{dup} \cdot \Pi)^*$. Then, unfolding $\text{accept}(g^{(b)}, y_1 \diamond \ldots \diamond y_n)$ yields

$$\begin{aligned} & \text{accept}(g^{(b)}, y_1 \diamond \ldots \diamond y_n) \\ \iff \quad & \exists g'.\, \tilde{D}(g^{(b)})(\alpha) = (\gamma, g' \cdot g^{(b)}) \wedge \text{accept}(g' \cdot g^{(b)}, \gamma \cdot z \diamond y_{k+1} \diamond \ldots \diamond y_n) \quad \text{(def.)} \end{aligned}$$

We fully evaluate $\tilde{D}(g')(\gamma)$ to get some output $\delta$ (which must occur by liveness of $g'$, where by "fully" we mean repeatedly applying $\tilde{D}$ on outputs $(\beta, e)$ like $\tilde{D}(e)(\beta)$ until $\tilde{D}(e)(\beta) \in \text{At}$). Then, let $\delta \cdot w$ be the remainder of the input string after $g'$ consumes its input. By the IH, $\delta \cdot w \in l^{\tilde{D}}(g^{(b)})$, so $\gamma \cdot z \diamond y_{k+1} \diamond \ldots \diamond y_n \in l^{\tilde{D}}(g' \cdot g^{(b)})$ and $t = y_1 \diamond \ldots \diamond y_n \in l^{\tilde{D}}(g^{(b)})$.

For the reverse direction $t = \alpha \cdot \pi_\beta \cdot \mathbf{dup} \cdot t' \in l^{\tilde{D}}(g^{(b)})$, then by definition $\exists g'.\, \tilde{D}(g^{(b)})(\alpha) = (\beta, g' \cdot g^{(b)})$ such that accept$(g' \cdot g^{(b)}, \beta \cdot t')$. Then, accept$(g' \cdot g^{(b)}, \beta \cdot t')$ if and only if there exists $x, y \in \mathrm{GT}$ for which $\beta \cdot t' = x \diamond y$ and accept$(g', x)$ and accept$(g^{(b)}, y)$. Since $D(g^{(b)})(\alpha) = (\beta, g' \cdot g^{(b)})$ iff $D(g)(\alpha) = (\beta, g')$, we know accept$(g', x)$ iff accept$(g, \alpha \cdot \pi_\beta \cdot \mathbf{dup} \cdot \pi_\beta \diamond x)$. Then

$$\begin{aligned}
\alpha \cdot \pi_\beta \cdot \mathbf{dup} \cdot \pi_\beta \diamond x \in [\![g]\!] \quad &\iff \quad \text{accept}(g, \alpha \cdot \pi_\beta \cdot \mathbf{dup} \cdot \pi_\beta \diamond x) && \text{(IH on } g\text{)} \\
&\iff \quad \text{accept}(g', x) && (D(g)(\alpha) = (\beta, g')) \\
&\iff \quad x \in [\![g']\!] && \text{(IH on } \beta \cdot t' = x \diamond y\text{)}
\end{aligned}$$

Also by the IH on $\beta \cdot t' = x \diamond y$ we know that accept$(g^{(b)}, y)$ iff $y \in [\![g^{(b)}]\!]$, so $\alpha \cdot \pi_\beta \cdot \mathbf{dup} \cdot t' \in l^{\tilde{D}}(g^{(b)})$ implies accept$(g' \cdot g^{(b)}, \beta \cdot t')$ for some $g'$, which implies accept$(g', x)$ and accept$(g^{(b)}, y)$ for whichever $x, y \in \mathrm{GT}$ have $\beta \cdot t' = x \diamond y$, which implies $x \in [\![g']\!]$ and $y \in [\![g^{(b)}]\!]$ by IHs, which implies $x \diamond y \in [\![g' \cdot g^{(b)}]\!]$, which implies $t = \alpha \cdot \pi_\beta \cdot \mathbf{dup} \cdot \pi_\beta \diamond x \diamond y \in [\![g^{(b)}]\!]$. We are done. □

## BIBLIOGRAPHY


[1] Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. Netkat: Semantic foundations for networks. *SIGPLAN Not.*, 49(1):113–126, jan 2014.

[2] Clemens Grabmayer and Wan Fokkink. A complete proof system for 1-free regular expressions modulo bisimilarity. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '20, page 465–478, New York, NY, USA, 2020. Association for Computing Machinery.

[3] Michael Greenberg, Ryan Beckett, and Eric Campbell. Kleene algebra modulo theories: a framework for concrete KATs. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. ACM, jun 2022.

[4] John Hopcroft and Richard Karp. A linear algorithm for testing equivalence of finite automata. *Cornell University Technical Report*, TR:71-114, 1971.

[5] D. Kozen. A completeness theorem for kleene algebras and the algebra of regular events. *Information and Computation*, 110(2):366–390, 1994.

[6] Dexter Kozen. Kleene algebra with tests. *ACM Trans. Program. Lang. Syst.*, 19(3):427–443, may 1997.

[7] J.J.M.M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249(1):3–80, 2000. Modern Algebra.

[8] Arto Salomaa. Two complete axiom systems for the algebra of regular events. *J. ACM*, 13(1):158–169, jan 1966.

[9] Todd Schmid, Tobias Kappé, Dexter Kozen, and Alexandra Silva. Guarded kleene algebra with tests: Coequations, coinduction, and completeness. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[10] Todd Schmid, Tobias Kappé, and Alexandra Silva. A complete inference system for skip-free guarded kleene algebra with tests, 2023.

[11] Steffen Smolka, Nate Foster, Justin Hsu, Tobias Kappé, Dexter Kozen, and Alexandra Silva. Guarded kleene algebra with tests: Verification of uninterpreted programs in nearly linear time. *CoRR*, abs/1907.05920, 2019.

[12] Robert Endre Tarjan. Efficiency of a good but not linear set union algorithm. *J. ACM*, 22(2):215–225, apr 1975.