Systematic Verification of Safety Properties of Arbitrary Network Protocol Compositions Using CHAIN[*]

Adam D. Bradley, Azer Bestavros, and Assaf J. Kfoury
Department of Computer Science, Boston University
artdodge,best,kfoury@cs.bu.edu

Abstract

*Formal correctness of complex multi-party network protocols can be difficult to verify. While models of specific fixed compositions of agents can be checked against design constraints, protocols which lend themselves to arbitrarily many compositions of agents–such as the chaining of proxies or the peering of routers–are more difficult to verify because they represent potentially infinite state spaces and may exhibit emergent behaviors which may not materialize under particular fixed compositions. We address this challenge by developing an algebraic approach that enables us to reduce arbitrary compositions of network agents into a behaviorally-equivalent (with respect to some correctness property) compact, canonical representation, which is amenable to mechanical verification. Our approach consists of an algebra and a set of property-preserving rewrite rules for the Canonical Homomorphic Abstraction of Infinite Network protocol compositions (*CHAIN*). Using* CHAIN*, an expression over our algebra (*i.e.*, a set of configurations of network protocol agents) can be reduced to another behaviorally-equivalent expression (*i.e.*, a smaller set of configurations). Repeated applications of such rewrite rules produces a canonical expression which can be checked mechanically. We demonstrate our approach by characterizing deadlock-prone configurations of HTTP agents, as well as establishing useful properties of an overlay protocol for scheduling MPEG frames, and of a protocol for Web intra-cache consistency.*

1. Introduction

Increasingly, the Internet is being used as a ubiquitous infrastructure supporting a multitude of distributed applications and services. The introduction, deployment, and revision of Internet services is laden with uncertainties that arise from our inability to formally establish the safety of such services–namely, that such services will not interfere with existing services or with prior versions of the same service. In light of the increasingly important role Internet-based services play in today's economy and society, it is incumbent upon the networking research community to develop sound formalisms by which such properties can be assessed, and to promote widespread use thereof. Indeed, in a recent NSF PI meeting of over 250 network researchers, improving the trustworthiness of Internet applications [25] and the development of formalisms to scale our understanding of networked systems [24] were deemed to be two of the most pressing challenges facing the networking community for the next few years.

Current efforts to assess the trustworthiness of the Internet have focused on proving desirable properties of an *individual* protocol or service agent, with very few efforts focusing on properties that emerge from the *composition* of such protocols and services (*e.g.*, [12]). Properties that emerge from the composition of similar yet distinct protocols and agents are much harder to reason about–not to mention check mechanically–due to the arbitrary nature of such compositions.

An illustrative example can be found in the networking community's experience with specifying a revised HTTP protocol standard, HTTP/1.1. While the original formulations of the HTTP protocol were truly stateless and thus deadlock-free, the addition of the `100 Continue` mechanism to HTTP/1.1 [17] introduced multiple states to the transaction model for clients, servers, and intermediaries; unfortunately, an ambiguity was found in the specification with respect to particular compositions of proxies of one protocol version with clients and servers of another, such that deadlocks could occur between "correct" implementations of HTTP/1.1 (RFC2068) and HTTP/1.0 [22].

Stateful multi-party protocols are notoriously difficult to get right. For years, analogous problems have been commonplace in the design of lower-level distributed protocols; mastering the nuances of handshaking, rendezvous, mutual exclusion, leader election, and flow control so as to guarantee correct, deadlock-free, work-accomplishing behavior requires very careful thought, and hardening the specifications and implementations of these protocols to deal with misbehaving or potentially hostile peers remains a difficult problem at all layers of the stack.

In this paper, we show how to use an algebraic approach coupled with traditional correctness-checking mechanisms like *model checkers* to systematically discover compositions of networked agents which give rise to problematic behav-

[*]This research was supported in part by NSF (awards ANI-9986397, ANI-0095988, CCR-9988529, ITR-0113193, ANI-020205294, and EIA-0202067) and U.S. Department of Education (GAANN Fellowship)

iors. This approach generalizes to characterizing a wide array of protocol correctness problems, including the HTTP issue discussed above.

Paper Contributions and Overview This paper proposes a systematic approach to the verification of safety properties in arbitrary compositions of network protocols using CHAIN–an algebra and associated rewrite rules for the *C*anonical *H*omomorphic *A*bstraction of *I*nfinite *N*etwork protocol compositions. We instantiate our approach for a number of network protocols, showing how it enables us to identify safety violations of *arbitrarily large compositions* of these protocols mechanically, using readily-available model checking technologies.

Within CHAIN, every type of agent within a composible protocol system (*e.g.,* "HTTP/1.0 server", "MPEG router", *etc.*) is represented by a single symbol; compositions of agents of these types are represented algebraically as strings ("chains") of those symbols. Protocols supporting arbitrarily composible intermediaries (*e.g.,* HTTP proxies) are described by an infinite set of such chains. We then develop a set of *rewrite rules* over this algebra which preserve the property under consideration. By repeated application of these rewrite rules to the set of all meaningful chains, we *reduce* this set to a much smaller canonically representative set of chains–an abstraction of the original set. This abstraction is a *homomorphic image* of the original set in the sense that if a property holds for the abstraction, then it provably holds for *all* of the possibly infinite compositions allowed by the protocol. Where a homomorphic image of finite size is derivable, the abstraction can then be exhaustively verified mechanically using off-the-shelf tools or methods.

The remainder of this paper is organized as follows. As a motivation for and a case study in the application of our methodology, we begin in Section 2 by outlining the HTTP request continuation mechanism, a feature of the HTTP/1.1 protocol, and its problems. We follow that in Section 3 with a presentation of the underpinnings of our CHAIN algebra and reductions. In Section 4 we bring the formalisms in CHAIN to bear on three examples of protocol compositions. First, we use it to characterize possible safety violations (deadlock scenarios) of HTTP/1.1 protocol compositions; we do so by translating previously established "equivalence" relationships into algebraic rewrite rules, thus creating a finite homomorphic image of the infinite set of HTTP compositions, allowing us to exhaustively identify the infinite sets of deadlock-prone and deadlock-safe compositions. Next, we illustrate the application of CHAIN to an MPEG packet routing protocol for overlay networks and a web intra-cache consistency protocol We conclude the paper in Section 6 with a summary and a brief discussion of future directions of this work.

2. HTTP Request Continuation

In HTTP/1.0, all transactions had a very simple and stateless communication model: (1) A client would send a whole request, *i.e.,* a request line, a set of headers, and an optional request entity; (2) The server, after receiving the whole request, would respond with a complete request, *i.e.,* a status line, a set of headers, and an optional response entity.

One of the desired features for HTTP/1.1 was the ability for clients to avoid transmitting very large entities with their requests when the transactions would fail independent of the content of the document (*e.g.,* an authentication failure or transient server problem) [17]. The original HTTP/1.1 specification (RFC2068) supports this capability by allowing clients to pause part-way through sending a request; the server may send an error code immediately, informing the client that the request has already failed and the remainder of the request should not be sent, or may send a `100 Continue` response, which tells the client to send the remainder of the request.

While the original specification of this mechanism was clearly sound with respect to simple client-server cases, it was ambiguous as to the correct behavior of proxies; compelling arguments were made that the RFC's language suggested both hop-by-hop and end-to-end interpretations of the feature. It was realized that, under at least one of these interpretations, certain combinations of correctly implemented components in the client-proxy-server chain were prone to deadlock [22]; an attempt at addressing this problem was made in the next public revision (RFC2616) with the introduction of the `Expect` mechanism and the clarification of the semantics of `100 Continue` with respect to proxies. Given that many existing implementations conformed to the various interpretations of RFC2068, it was decided that RFC2616 should also include a number of heuristics to facilitate graceful interoperation with those implementations. The resulting quagmire of special-case interoperability rules and the set of possible combinations of revisions in the various roles makes it difficult to say anything with certainty about the correctness and full interoperability of the specification; while it seemed *reasonably* (and even *empirically*) to be correct, it was not *provably* so.

In previous work [5], we presented a set of models for HTTP clients, proxies, and servers. Any single combination of a client, some proxies, and a server (hereafter an *arrangement*) can be examined using a finite-state modeling tool like SPIN [14] which instantiates the models and joins them with message channels to determine whether any possible execution of *that* arrangement can lead to an undesirable state (*e.g.,* deadlock or livelock); by strategically selecting a set of interesting arrangements, we were able to elaborate upon the community's understanding of the problem.

Brute-force assessment the *interoperability* of the modeled agents in all possible client-proxy-server arrangements, however, would have required us to verify $|\mathcal{C}| \times |\mathcal{S}| \times \left(\sum_{i=0}^{\infty} |\mathcal{P}|^i\right)$ arrangements, where $\mathcal{C}$ is the set of client models, $\mathcal{P}$ is the set of proxy models, and $\mathcal{S}$ is the set of server models, in order to examine all possible interactions between client, proxy, and server behaviors. Using this brute-force approach, not only would a "complete" proof require verifying an infinite number of arrangements, but even a "partial" proof (all cases up to N instead of ∞ proxies) requires verifying a number of arrangements exponential in N.

We previously established that particular arrangements are provably equivalent to others in terms of their externally-observable behaviors [5]. With a sufficient set of such relationships, one could potentially reduce an arbitrarily large set of arrangements to a much smaller (preferably finite) set of behaviorally representative arrangements, which could then plausibly be verified individually. To be useful, the discovery *and* application of such reductions must follow a systematic approach. Much work has already been done in several communities on discovery of such equivalence relationships (*e.g.*, [8, 2, 20]); in the remainder of this paper, we presume such results and develop the *systematic strategy* for model space reduction which they enable.

It is also important to note that a number of techniques already exist for checking models with infinite state spaces (*e.g.*, [15, 10, 19, 9, 18]). These techniques tend (in their current state) to be fairly opaque in the sense that they often cannot connect their internal representations of the infinite state space with intuitively useful descriptions of behaviors of particular protocol agents. For those designing and developing distributed protocols, we believe that the ability to do so is crucially important to facilitate debugging of flawed protocols.

3. The CHAIN Approach

In this section we present the details of our algebraic approach, CHAIN (a system for the *C*anonical *H*omomorphic *A*bstraction of *I*nfinite *N*etwork protocol compositions). Intuitively, CHAIN represents protocol compositions using *strings* (chains); the infinite set of such chains is reduced to a representative finite set via reduction relations (*rewrite rules*) which preserve some correctness property. This section discusses the formal structure and properties of these components which give rise to the desirable properties of a CHAIN system (correct abstraction, sufficient expressive power, homomorphism, termination, canonicity of result).

3.1. Arrangements in CHAIN

Let $G = (\mathcal{N}, \mathcal{E})$ be a graph where the members of $\mathcal{N}$ denote the types of agents which will make up our models and $\mathcal{E}$ are directed edges which indicate valid sequences of those types of agents (that is, if there is an edge from node n_1 to n_2, then n_2 may immediately follow n_1 in a composition). The set of *chains* in G is then simply the set of finite paths in G, which we denote $\mathsf{paths}(G)$.

Notice that our definition of *chains* includes sequences which will make up "partial" network setups, *e.g.*, client connected with a series of proxies (but no server), or even an empty sequence. For this reason, we also define the set of *arrangements* as $\mathcal{A} \subseteq \mathsf{paths}(G)$ such that $\mathcal{A}$ are the maximal-length members of $\mathsf{paths}(G)$; for example, all members of $\mathcal{A}$ for the HTTP application are chains beginning with a client followed by a chain of zero or more proxies and terminating with a server.

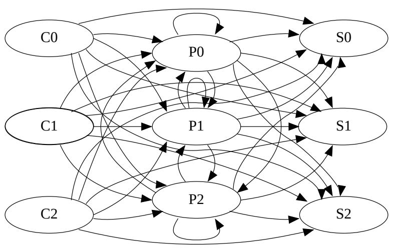

Figure 1. G for HTTP arrangements ($\mathcal{A}$)

Table 1. HTTP agent models

Standard	Client	Proxy	Server
RFC1945 (HTTP/1.0)	C0	P0	S0
RFC2068 (obsolete HTTP/1.1)	C1	P1	S1
RFC2616 (HTTP/1.1)	C2	P2	S2

As an example, for the HTTP protocol, G would be Figure 1, where models of clients are represented as Cn (n identifies the protocol revision per Table 1); similarly, proxies are represented by Pn, servers by Sn.

3.2. Arrangement Properties

We are interested in identifying the members of $\mathcal{A}$ which satisfy (or fail to satisfy) desirable properties, *e.g.*, those that are *deadlock-free*. Let π denote such a property, which can be viewed as a boolean function $\pi : \mathcal{A} \to \{\textbf{true}, \textbf{false}\}$.

The primary methodological goal of CHAIN is to obtain a "friendly" specification of the two sets:

$$\begin{aligned} \mathcal{A}_{\textbf{true}} &= \{a \in \mathcal{A} \,|\, \pi(a) = \textbf{true}\} \text{ and} \\ \mathcal{A}_{\textbf{false}} &= \{a \in \mathcal{A} \,|\, \pi(a) = \textbf{false}\} \,. \end{aligned}$$

By "friendly" we mean, at a minimum, there is a feasible computation to determine whether $a \in \mathcal{A}_{\textbf{true}}$ or $a \in \mathcal{A}_{\textbf{false}}$ for any arrangement (no matter how long) in $\mathcal{A}$; ideally, this should take the form of descriptions of the sets $\mathcal{A}_{\textbf{true}}$ and $\mathcal{A}_{\textbf{false}}$ which can be used to quickly (in polynomial time or better) test whether $a \in \mathcal{A}_{\textbf{true}}$ or $a \in \mathcal{A}_{\textbf{false}}$.

3.3. CHAIN Reductions

Consider a graph G as described above and a property π on the set $\mathcal{A}$ of arrangements in G. We denote the powerset of a set S by 2^S. We extend $\pi : \mathcal{A} \to \{\textbf{true}, \textbf{false}\}$ to a function $\pi : 2^{\mathcal{A}} \to \{\textbf{true}, \textbf{false}\}$ by defining for every $A \in 2^{\mathcal{A}}$:

$$\pi(A) = \begin{cases} \textbf{true} & \text{if } \pi(a) = \textbf{true} \text{ for every } a \in A, \\ \textbf{false} & \text{if } \pi(a) = \textbf{false} \text{ for some } a \in A. \end{cases}$$

Let $\mathcal{A}'$ be some subset, not necessarily proper, of the set $\mathcal{A}$ of arrangements in G. Because $\mathcal{A}$ is a subset of $\mathsf{paths}(G)$, so is $\mathcal{A}'$ a subset of $\mathsf{paths}(G)$. A *reduction function on* $\mathcal{A}'$ is a function $f : \mathsf{paths}(G) \to 2^{\mathsf{paths}(G)}$ satisfying two conditions:

Invariance on $\mathcal{A}'$: For every $a \in \mathcal{A}'$, it is the case that $\pi(f(a))$ is defined and $\pi(f(a)) = \pi(a)$.

Progress on $\mathcal{A}'$: $\left(\bigcup_{a\in\mathcal{A}'} f(a)\right) \subsetneq \mathcal{A}'$.

We can extend $f : \mathsf{paths}(G) \to 2^{\mathsf{paths}(G)}$ to a function $f : 2^{\mathsf{paths}(G)} \to 2^{\mathsf{paths}(G)}$ by setting $f(A) = \bigcup_{a\in A} f(a)$ for every $A \in 2^{\mathsf{paths}(G)}$. Thus, the *progress* condition above can be expressed more succinctly as $f(\mathcal{A}') \subsetneq \mathcal{A}'$.

Informally, the *invariance* condition says that π is an invariant of the transformation from $a \in \mathcal{A}'$ to $f(a) \subset \mathcal{A}'$. In practice, this means that, in order to test whether $a \in \mathcal{A}'$ satisfies property π, it suffices to test whether every $b \in f(a)$ satisfies π; as a rule, a desirable reduction is one in which the aggregate of the latter tests is "easier" computationally than the former test.

The *progress* condition is assurance that we gain something by carrying out the transformation from $a \in \mathcal{A}'$ to $f(a) \subset \mathcal{A}'$, *i.e.*, the set $f(\mathcal{A}')$ is a non-empty proper subset of $\mathcal{A}'$. In practice, should $\mathcal{A}'$ be an infinite set we will also want $\mathcal{A}' - f(\mathcal{A}')$ to be an infinite set, *i.e.*, infinitely many arrangements are excluded from the search space $\mathcal{A}'$ by the reduction.

The key insight behind a reduction is that it establishes *behavioral equivalence with respect to* π within some set of chains; a reduction is a statement that "the behaviors of members of set $\mathcal{A}'$ are fully represented by the behaviors of members of its subset $f(\mathcal{A}')$". The means by which this behavioral equivalence is established may be any mechanism appropriate to the given π (*e.g.*, logical proofs, type systems [8], process algebra [2], theory of I/O automata [20], I/O equivalence, *etc.*).

3.4. Reduction Strategy

Intuitively, our strategy is to identify a set of reductions (*i.e.*, congruence relations over $\mathcal{A}$ which preserve behavioral equivalence) by which we can establish a finite-sized homomorphic image of $\mathcal{A}$ (that is, a finite-sized $\mathcal{A}_n \subset \mathcal{A}$ such that the behaviors of every member of $\mathcal{A}$ correpond with behaviors of corresponding members of $\mathcal{A}_n$).

Starting from $\mathcal{A}_0 = \mathcal{A}$, our proposed strategy is to define a nested sequence of strictly decreasing subspaces:

$$\mathcal{A}_0 \supset \mathcal{A}_1 \supset \cdots \supset \mathcal{A}_n$$

induced by a sequence of appropriately defined functions $g_1, g_2, \ldots, g_n$ where $g_i : \mathsf{paths}(G) \to 2^{\mathsf{paths}(G)}$ is derived from a reduction function on $\mathcal{A}_{i-1}$ and $\mathcal{A}_i = g_i(\mathcal{A}_{i-1})$ for every $1 \leqslant i \leqslant n$. With a sufficient set of reductions, this strategy produces a finite search space $\mathcal{A}_n$ such that

$$\mathcal{A}_n = g_n(\cdots(g_2(g_1(\mathcal{A})))\cdots)$$

which implies that for every $a \in \mathcal{A}$

$$\begin{array}{rcl} \mathcal{A}_n & \supseteq & g_n(\cdots(g_2(g_1(a)))\cdots) \text{ and} \\ \pi(a) & = & \pi(g_n(\cdots(g_2(g_1(a)))\cdots)) . \end{array}$$

3.5. Practical Specification of Reductions

We introduce a particular notion of *rewrite rules*. Each such rewrite rule R will be specified by an expression of the form

$$R: \ X \ \rhd \ \{Y_1, \ldots, Y_n\}$$

where $X, Y_1, \ldots, Y_n$ are each strings of agent names and variable names. Let $a, b_1, \ldots, b_n \in \mathsf{paths}(G)$. We say a *rewrites to the set* $\{b_1, \ldots, b_n\}$, using rule R in one step, which we express as:

$$a \rhd_R \{b_1, \ldots, b_n\}.$$

A rewrite rule R as described above induces a function $f_R : \mathsf{paths}(G) \to 2^{\mathsf{paths}(G)}$ as follows. For every $a \in \mathsf{paths}(G)$, we define:

$$f_R(a) = \begin{cases} \{a\} & \text{if } a \not\rhd_R B \text{ for all finite } B \subset \mathsf{paths}(G), \\ \bigcup\{B \subset \mathsf{paths}(G) \mid a \rhd_R B\} & \text{otherwise.} \end{cases}$$

Following standard notation, we write $f_R^{(0)}(a) = \{a\}$ and $f_R^{(k+1)}(a) = f_R(f_R^{(k)}(a))$ for all $k \geqslant 0$. We also define the function $f_R^{(*)} : \mathsf{paths}(G) \to 2^{\mathsf{paths}(G)}$ as follows. For every $a \in \mathsf{paths}(G)$:

$$f_R^{(*)}(a) = \begin{cases} f_R^{(k)}(a) & \text{if there exists } k \geqslant 0 \text{ such that} \\ & \quad f_R^{(k+1)}(a) = f_R^{(k)}(a), \\ & \quad \text{where } k \text{ is the least such,} \\ \text{undefined} & \text{if no such } k \geqslant 0 \text{ exists.} \end{cases}$$

Informally, $f_R^{(*)}(a)$ returns a fix-point of f_R obtained by repeated application of f_R to a, if such a fix-point exists.

Now, consider the set $\mathcal{A}$ of arrangements in G, a property π on $\mathcal{A}$, and some subset $\mathcal{A}' \subseteq \mathcal{A}$. We say that the rewrite rule R is a *reduction on* $\mathcal{A}'$ provided that the function $f_R^{(*)} : \mathsf{paths}(G) \to 2^{\mathsf{paths}(G)}$ induced by R is a *reduction on* $\mathcal{A}'$ satisfying the two conditions defined in Section 3.3: *invariance* on $\mathcal{A}'$ and *progress* on $\mathcal{A}'$.

Our rewrite rules will satisfy a pleasant condition guaranteeing that $f_R^{(*)}(a)$ is always defined. Let us say that the rule R is *bounded-monotonic in* M *iff* for some metric M with a minimum value and for all $a, b_1, \ldots, b_n \in \mathsf{paths}(G)$

such that $a \rhd_R \{b_1, \ldots, b_n\}$,

$$\mathsf{M}(a) > \mathsf{M}(b_1)\ ,\ \ldots\ ,\ \mathsf{M}(a) > \mathsf{M}(b_n).$$

Lemma 3.1. *If the rewrite rule R is bounded-monotonic then, for every $a \in \mathsf{paths}(G)$, it holds that $f_R^{(*)}(a)$ is defined, and is a non-empty finite subset of $\mathsf{paths}(G)$.*

The simplest choice for such a metric is string length (which has a minimum value of zero); where a rule is not *length-decreasing*, it must be shown to be bounded-monotonic in some other metric to ensure $f_R^{(*)}$ is defined.

3.6. Confluence of Chain Reductions

As with any term re-writing system, at least two properties are important to establish: *termination* (*i.e.*, every arrangement can be rewritten only finitely many times) and *confluence* (*i.e.*, if a can be rewritten to b_1 and b_2, then further rewriting of both of those can produce a single string c). Termination ensures that the system draws some conclusion, while confluence demonstrates the system's internal consistency; the combination of these two properties implies that the final result of the rewriting process is canonical, and that the rewriting process encodes an *equivalence class* for each possible result. Establishing confluence also helps to minimize the size of the result set.

A set of rewrite rules terminate if the effects of all rules are *bounded-monotonic* with respect to a single metric; for example, if all rules are *length-decreasing* then a system of such rules must terminate. By Newman's Lemma [1], we know that a rewrite system which terminates is confluent *iff* it is locally confluent (*i.e.*, given arrangement a which can be rewritten in one step to B_1 or to B_2, both B_1 and B_2 can be rewritten in zero or more steps to some B_3). We therefore need only determine local confluence between pairs of rewrite rules in order to establish confluence.

In the simplest cases, a pair of rewrite rules R_i and R_j are locally confluent if they are commutative, *i.e.*, if $f_i^{(*)}(f_j^{(*)}(a)) \equiv f_j^{(*)}(f_i^{(*)}(a))$ for all $a \in \mathcal{A}$. As much as possible, we use this property in our confluence proofs.[1]

It should be noted that a non-confluent set of rewrite rules is not a failure of our methodology. In systems like ours where rewrite rules represent *equivalence*, that equivalence is transitive (regardless of the directionality of the rewrite); it follows that any such divergence identifies additional equivalence relationships which themselves embody additional valid rewrite rules. As such, any arrangement $a \in \mathcal{A}$ for which two rewrite rules R_i and R_j provide diverging evaluation paths actually becomes an instance of a proper behavioral equivalence relation; $f_i(a)$ and $f_j(a)$ are behaviorally equivalent sets (because of the *Invariance* property), and as such if either set has only one member, the pair can be directly transformed into a previously unknown reduction. Such additional relations resolve the failures of local confluence, but must then be tested for interference with each other and the original rule set. The whole process can be mechanized with the Knuth-Bendix procedure [16].

[1]This is a special case of the general definition of local confluence: A pair is locally confluent if there exist compositions (call them F_1 and F_2) of f_is drawn from the full set of valid rewrite rules ($\mathcal{R}$) such that $F_1(f_i^{(*)}(a)) \equiv F_2(f_j^{(*)}(a))$ for all $a \in \mathcal{A}$. Intuitively, this means that systems of rules are confluent when a divergence in rewriting strategies can be reconciled by some sequences of additional rewrites.

3.7. Homomorphic Images

If $f_R^{(*)}$ is always defined, then the application of $f_R^{(*)}$ to all members of a set $\mathcal{A}$ will yield some subset of $\mathcal{A}$ such that, for every $a \in \mathcal{A}$, the value of $\pi(a)$ can be easily determined from $\pi(f_R^{(*)}(a))$. Thus, $f_R^{(*)}(\mathcal{A})$ is a *homomorphic image* of $\mathcal{A}$.

In the rest of the paper, when there is no ambiguity, notions that have been defined for a rewrite rule R are extended to the function $f_R^{(*)}$ in the obvious way; for example, we say "$f_R^{(*)}$ is length-decreasing" if R is length-decreasing.

It is also convenient to introduce the notion of the *support* of the function $f_R^{(*)}$, or of its associated rewrite rule R:

$$\mathsf{support}(f_R^{(*)}) \ =\ \mathsf{support}(R) \ =\ \{a \in \mathcal{A} \,|\, f_R^{(*)}(a) \neq \{a\}\},$$

i.e., $\mathsf{support}(f_R^{(*)})$ is the portion of $\mathcal{A}$ on which $f_R^{(*)}$ acts non-trivially; so,

$$f_R^{(*)}(\mathcal{A}) = (\mathcal{A} - \mathsf{support}(f_R^{(*)})) \cup f_R^{(*)}(\mathsf{support}(f_R^{(*)}))$$

Recall our stated strategy from Section 3.4: to define a nested sequence of strictly decreasing subspaces $\mathcal{A} = \mathcal{A}_0 \supset \mathcal{A}_1 \supset \cdots \supset \mathcal{A}_n$ induced by a sequence of reduction functions $f_1, \ldots, f_n$. In what follows, for every $1 \leqslant i \leqslant n$, we use f_i to denote the $f_{R_i}^{(*)}$ induced by the bounded-monotonic rewrite rule R_i.

A trivial approach to this goal is to find $f_n(\cdots(f_2(f_1(\mathcal{A})))\cdots)$. However, such a formulation fails to properly account for *convolutions* of rewrite rules. For example, consider a system in which R_1 reduces all sequences of "a"s to a single "a" and R_2 removes any "b" appearing immediately between two "a"s; for the string aba, clearly $f_2^{(*)}(f_1^{(*)}(\mathsf{aba})) \neq f_1^{(*)}(f_2^{(*)}(f_1^{(*)}(\mathsf{aba})))$; the convolution of f_1 and f_2 has a greater effect than their sequential application.

Rather than composing the functions as such, we individually consider the effect of each rule upon $\mathcal{A}$ as a whole, and then take the intersection of the resulting homomorphic images. The intersection operator allows our examination of each f_i to wholly exclude from future consideration all arrangements "reduced away" by each f_i.

Lemma 3.2. *Consider a set of reductions $\mathcal{R} = \{R_1, \ldots, R_n\}$ inducing functions $\mathcal{F} = \{f_{R_1}^{(*)}, \ldots, f_{R_n}^{(*)}\}$ which are all monotonic with respect to some metric. The* *homomorphic image $\mathcal{A}_n$ can then be defined as:*

$$\mathcal{A}_n = \bigcap_{i=1}^{n} \left(f_{R_i}^{(*)}(\mathcal{A}) \right)$$

which can be equivalently stated as:

$$\mathcal{A}_n \;=\; \bigcap_{i=1}^{n} \left(\mathcal{A} - \mathsf{support}(f_{R_i}^{(*)}) \right) \cup f_{R_i}^{(*)}(\mathsf{support}(f_{R_i}^{(*)})) \; .$$

Where the set of reductions is clear from context and where $\mathcal{A}_n$ (the smallest homomorphic image supported by the given reductions) is of finite ordinality, we call it $\mathcal{A}_\top$.

It follows then that $\pi(\mathcal{A}) = \pi(\mathcal{A}_\top)$, *i.e.*, if some property (*e.g.*, freedom from deadlocks) is provable for all members of the minimal homomorphic image ($\{\pi(a) = \textbf{true} \,|\, a \in \mathcal{A}_\top\}$), then it must also hold for all members of the (infinite) set $\mathcal{A}$. More generally, $\{a \in \mathcal{A}_\top \,|\, \pi(a) = \textbf{true}\}$ is the homomorphic image of $\mathcal{A}_{\textbf{true}}$, and $\{a \in \mathcal{A}_\top \,|\, \pi(a) = \textbf{false}\}$ is the homomorphic image of $\mathcal{A}_{\textbf{false}}$; therefore, finding $\mathcal{A}_\top$ essentially achieves the core methodological goal of CHAIN as stated in Section 3.2.

4. Example Applications of CHAIN

In this section, we exemplify the application of the CHAIN approach to a series of interesting network protocol correctness problems. We begin with a detailed completion of our example of HTTP request continuation deadlock-safety in order to clearly illustrate the workings of CHAIN. We then proceed to a more abbreviated discussion of its application to the analysis of an applet for selective dropping of MPEG frames in an overlay network. Finally, we sketch its application to a web intra-cache consistency algorithm.

4.1. HTTP Deadlock-Safety

Through careful study of our models of HTTP protocol agents, we have derived and proven a set of rewrite rules which preserve the behavior of chains with respect to HTTP request continuation. If an arrangement is deadlock-prone, then any arrangement which can be rewritten to that one will also be deadlock-prone; likewise, any arrangement to which it can be rewritten will also be deadlock-prone. The same holds for arrangements which are deadlock-free. The derivation of these and other rules is discussed in greater depth in [6]. Eight of the rules derived there pertaining to our current goal are presented in Table 2, along with two more rules (R_9 and R_{10}), the derivation of which will be discussed below.

In this paper, we refer to particular models using the letter-number pairs presented in Table 1; these represent the favored models for each revision/role.

All of the listed rules are proper rewrite rules as defined in Section 3.5. Notice that these rules are also all length-decreasing, which implies (by Lemma 3.1) that $f_{R_i}^{(*)}$ is always defined for all of them. Therefore, each of the preceding rewrite rules R_i gives rise to a function $f_{R_i}^{(*)}$, henceforth denoted by f_i.

Table 2. Rewrite rules $R_1, \ldots, R_{10}$ and resulting homomorphic images $f_1(\mathcal{A}), \ldots, f_{10}(\mathcal{A})$

	Rewrite Rule	A_i, *i.e.*, $f_i(\mathcal{A})$
R_1	x P0 y $\rhd_{R_1}$ { x S0, C0 y }	$\mathcal{A} - \mathcal{CP}^*$ P0 $\mathcal{P}^*\mathcal{S}$
R_2	x P0 P0 y $\rhd_{R_2}$ { x P0 y, C0 S0 }	$\mathcal{A} - \mathcal{CP}^*$ P0 P0 $\mathcal{P}^*\mathcal{S}$
R_3	C2 P2 x $\rhd_{R_3}$ { C2 x }	$\mathcal{A} -$ C2 P2 $\mathcal{P}^*\mathcal{S}$
R_4	x P2 S2$\rhd_{R_4}$ { x S2 }	$\mathcal{A} - \mathcal{CP}^*$ P2 S2
R_5	x P2 P2 y $\rhd_{R_5}$ { x P2 y }	$\mathcal{A} - \mathcal{CP}^*$ P2 P2 $\mathcal{P}^*\mathcal{S}$
R_6	x P1 P1 P1 y $\rhd_{R_6}$ { x P1 P1 y }	$\mathcal{A} - \mathcal{CP}^*$ P1 P1 P1 $\mathcal{P}^*\mathcal{S}$
R_7	C0 P1 x $\rhd_{R_7}$ { C1 x }	$\mathcal{A} -$ C0 P1 $\mathcal{P}^*\mathcal{S}$
R_8	x P1 P2 P1 y $\rhd_{R_8}$ { x P1 P1 y }	$\mathcal{A} - \mathcal{CP}^*$ P1 P2 P1 $\mathcal{P}^*\mathcal{S}$
R_9	C1 P1 P1 x $\rhd_{R_9}$ { C1 P1 x }	$\mathcal{A} -$ C1 P1 P1 $\mathcal{P}^*$ $\mathcal{S}$
R_{10}	C1 P2 P1 x $\rhd_{R_{10}}$ { C1 P1 x }	$\mathcal{A} -$ C1 P2 P1 $\mathcal{P}^*$ $\mathcal{S}$

Notice also that each f_i (*i.e.*, each $f_{R_i}^{(*)}$) is a valid reduction function, in that it satisfies the *invariance* and the *progress* properties. Invariance was previously established; progress holds because for every f_i it is true that $f_i(\mathcal{A}) \subsetneq \mathcal{A}$.

4.1.1 Confluence

We next ask whether this set of reductions is confluent, *i.e.*, whether every arrangement $a \in \mathcal{A}$ will be terminally rewritten to a single result set independent of the reduction strategy. Because our set of rewrite rules will terminate (because they are all length-decreasing), this is equivalent to asking if the set of reductions is locally confluent.

The local confluence of most pairs of reductions is straightforward to see, because the rules are independent (*i.e.*, non-interfering) and therefore commutative. Clearly R_1 and R_2 operate upon chains which no other reductions operate upon; the cluster of R_3, R_4, and R_5 likewise are clearly independent in their effects of the predicate chains of R_1, R_2, R_6, R_7, and R_8, and similarly the cluster of R_6, R_7, and R_8 are independent of the first five rules. So our only concern is local confluence within these three clusters.

R_1 and R_2: Since R_2 is an instantiation of R_1, these two clearly do not lead to contradictory rewrite strategies.

R_3, R_4 and R_5: All three of these rules remove P2 from chains. Consider the one case where both R_3 and R_4 affect removal of the same P2, namely, a = C2 P2 S2. f_3 and f_4 remain commutative, because $f_{R_3}^{(0)}(f_{R_4}^{(1)}(a)) = f_{R_4}^{(0)}(f_{R_3}^{(1)}(a))$. Similarly, consider the set of chains over which R_3 and R_5 conflict, namely, a = C2 P2 P2 x for any x; f_3 and f_5 are clearly commutative because $f_{R_3}^{(0)}(f_{R_5}^{(1)}(a)) = f_{R_5}^{(0)}(f_{R_3}^{(1)}(a))$ when f_5 affects the specified subchain of a. The same proof holds for the pairing of R_4 and R_5. Since all three pairings of these three rewrites commute, the set is locally confluent.

R_6, R_7 and R_8: While rules R_6 and R_8 are clearly independent in their effects, the other two pairs within this cluster are not commutative.

- R_6 and R_7: These rewrite rules diverge on chains of the form C0 P1 P1 P1 x. R_7 rewrites this expression to C1 P1 P1 x, and R_6 rewrites it to C0 P1 P1 x; while the latter can then be rewritten using R_7 to C1 P1 x, they still identify different sets, and there exists no rewrite strategy which will (for all values of x) rewrite both of these to a common expression.
- R_7 and R_8: These rewrite rules diverge on chains of the form C0 P1 P2 P1 x. R_7 rewrites this expression to C1 P2 P1 x, while R_8 rewrites it to C0 P1 P1 x. Much as above, the second expression can again be re-written to C1 P1 x, but from there no rewrite strategy exists to rewrite these to a common expression.

Applying the procedure discussed in Section 3.6, each of these conflicts is transformed into a new valid rewrite rule in a straightforward way so as to preserve the length-decreasing property; the results are rules R_9 and R_{10}, which succeed in rendering the system confluent, so further iteration of the procedure is not necessary.

Theorem 4.1. *The set of rewrite rules $\mathcal{R} = \{R_1, \ldots, R_{10}\}$ is confluent.*

Proof. The local confluence of most rule pairs is already discussed above. Both of the failures of local confluence among $\{R_1, \ldots, R_8\}$ are addressed by application of the two new rewrite rules, R_9 and R_{10}. The two new rules are independent of each other and independent of the original eight rules, with the exception that R_9 and R_6 have identical effect upon expressions C1 P1 P1 P1 x and are thus commutative, and that R_{10} and R_6 have identical effect upon expressions C1 P2 P1 P1 P1 x and are thus commutative. Therefore, $\mathcal{R}$ is locally confluent.

Because the rules are all length-decreasing (and therefore bounded-monotonic), the system terminates; by Newman's Lemma, it is therefore confluent. □

4.1.2 Reducing the Model Space

Recall Lemma 3.2, which is the "glue" of our strategy. For each f_i we find $f_i(\mathcal{A})$, *i.e.*:

$$(\mathcal{A} - \mathsf{support}(f_i)) \cup f_i(\mathsf{support}(f_i))$$

We denote such a set induced by any f_i as A_i. Using the above-described supports and the sets they are mapped to, Table 2 presents these in simplified regular expression form for each of f_1 through f_{10}. For brevity, $\mathcal{C}$ = (C0 | C1 | C2), $\mathcal{P}$ = (P0 | P1 | P2), and $\mathcal{S}$ = (S0 | S1 | S2).

Taking the intersection of these homomorphic images gives us the finite minimal homomorphic image supported by the given reduction functions. We will use both regular expressions and finite state automata (in the style of the graph G described earlier) to describe such sets of strings for the remainder of this paper.

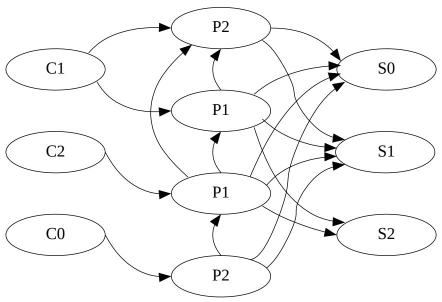

Figure 2. Automaton $\mathcal{A}_{10} - \mathcal{CS}$

The intersection of the ten sets $A_1 \ldots A_{10}$ is presented in Figure 2 as an automaton. (For visual clarity, the $\mathcal{CS}$ edges have been omitted; each C node also has an edge to each S node.) This represents the minimal homomorphic image of $\mathcal{A}$ under the ten reductions $R_1 \ldots R_{10}$. As such, we have satisfied the goal of our strategy by identifying a finite $\mathcal{A}_n = A_1 \cap \cdots \cap A_{10}$ which is a homomorphic image of $\mathcal{A}$. This set $\mathcal{A}_{10}$ has 29 member strings; thus, it is sufficient to compute $\pi(a)$ for only these 29 members of $\mathcal{A}$ in order to acquire a trivial procedure for the determination of $\pi(a)$ for any $a \in \mathcal{A}$. We have thus proven the following theorem:

Theorem 4.2. *Let $\mathcal{A}$ be the infinite space of all arrangements of HTTP agents as defined in Figure 1, and let $\mathcal{R}$ be the set of rewrite rules presented in Table 2. We can construct a finite subset $\mathcal{A}_\top$ of $\mathcal{A}$, consisting of* 29 *member arrangements, which satisfies the following condition: By the application of $\mathcal{R}$, every $a \in \mathcal{A}$ can be rewritten to a subset B of $\mathcal{A}_\top$ such that a satisfies π if and only if every $b \in B$ satisfies π.*

Proof. Follows directly from Lemma 3.2 and the above analysis. □

Theorem 4.3. *All $a \in \mathcal{A}$ such that $\pi(a) =$* **false** *will match one of the regular expressions:*

$$((\mathcal{C}\ \mathcal{P}^*\ \text{P1})\ |\ \text{C1})\ (\text{P1}\ |\ \text{P2}^+)\ (\text{S0}\ |\ \text{P0}\ \mathcal{P}^*\ \mathcal{S})$$
$$((\mathcal{C}\ \mathcal{P}^*\ \text{P1})\ |\ \text{C1}\ |\ \text{C2})\ (\text{P1}\ |\ \text{P2})^*\ \text{P1}\ (\text{S0}\ |\ \text{P0}\ \mathcal{P}^*\ \mathcal{S})$$

Proof. Among $a \in \mathcal{A}_{10}$, all a such that $\pi(a) =$ **false** (that is, all $a \in \mathcal{A}_{\textbf{false}}$) match at least one of the stated patterns, and no a such that $\pi(a) =$ **true** (*i.e.*, no $a \in \mathcal{A}_{\textbf{true}}$) matches either.

As we have shown, all members of $\mathcal{A}$ which are deadlock-prone are reducible to members of $\mathcal{A}_{10}$ which are deadlock-prone, and similarly, all members of $\mathcal{A}$ which are deadlock-safe are reducible to members of $\mathcal{A}_{10}$ which are deadlock-safe.

As such, the correctness of this theorem rests upon three properties: (1) for any member $a \in \mathcal{A}_{10}$, $\pi(a) =$ **false** *iff* a is in the union of these patterns (that is, the patterns correctly identify $\mathcal{A}_{\textbf{false}} \cap \mathcal{A}_{10}$); (2) any member of the union of these two patterns is reducible to a member of $\mathcal{A}_{\textbf{false}} \cap \mathcal{A}_{10}$; (3) no arrangement $a \in \mathcal{A}$ which does not match either of these patterns can be reduced to one which does.[2] Item-by-item proofs of these properties can be found in the Appendices of [7]. □

4.2. An MPEG Overlay Routing Protocol

While model checking is often applied in *post mortem* fashion to assess bugs and problems in existing protocols and software, this need not be the case. This section presents an application of CHAIN approaching the problem from a design perspective rather than a retroactive analysis perspective.

Many algorithms proposed for overlay networks are, by their nature, designed to be deployed into a network in which they will interact with other applications, as well as conventional routers and hosts (*i.e.*, we expect that they will be composed with other processes, perhaps both controlled and emergent). Thus it seems reasonable to believe that such applications are good candidates for analysis using CHAIN.

Consider the method for handling MPEG flows proposed in [13], which drops MPEG frames based upon its own drop history and the dependency and priority relationships between the three classes of MPEG frames (I, P, and B frames).[3] These relationships suggest simple packet-dropping rules: (1) If at all possible, dropping I frames should be avoided; (2) once a B frame has been dropped, all successor B frames until the next P frame are useless and should be dropped; and similarly (3) once a P frame has been dropped, all successor packets can be safely dropped until the next I frame. The applet presented in [13] implements a simple version of this algorithm requiring constant time and storage.

Unfortunately, the router[4] so described must itself receive all of the packets constituting an MPEG stream in order to behave correctly. This is an unreasonably optimistic assumption [23]; it is not hard to devise pathological reorderings among sequential pairs of packets which can cause the applet to wrongly treat large numbers of frames as "worthless" (and therefore discardable), and if certain packets are dropped before reaching such a router, it can erroneously forward large numbers of worthless packets.

We can easily cast either or both of these concerns (packet ordering and drop-tolerance) in CHAIN. There is nothing intrinsic to a network which drops or reorders packets that prevents it from being represented as an agent in the same sense in which the MPEG router is an agent. Therefore, it makes sense for us to ask within our framework whether the composition of a packet-reordering network with such a router would cause it to behave erroneously (that is, to drop packets which could still be valuable to an end-host)? Does the direct composition of two such routers induce packet drops that a single such router would not? What about composing two such routers using a packet-reordering network, or composing two network-router pairs? What about compositions with other kinds of routers which perform random drops, or which may do retransmissions on their own (*e.g.*, a wireless base station)? All of these questions can be framed in terms of a binary correctness property π which determines whether packets could be erroneously dropped by any particular composition of those components.

4.2.1 Representing and Reducing the Network

Some reductions naturally arise as properties of the basic existing network infrastructure; For example:

$$\begin{aligned} R_1 &: \quad x\ \textit{reord reord}\ y \ \triangleright\ \{x\ \textit{reord}\ y\} \\ R_2 &: \quad x\ \textit{drop drop}\ y \ \triangleright\ \{x\ \textit{drop}\ y\} \\ R_3 &: \quad x\ \textit{reord drop}\ y \ \triangleright\ \{x\ \textit{drop reord}\ y\} \end{aligned}$$

Notice that the third rule forces a particular ordering upon adjoining *reord* and *drop* nodes; in so doing, these three rules together form a compound rule which says that any chain consisting only of both *drop* and *record* can be represented with a single canonical chain, "*drop reord*". Excluding the inverse rule to R_3 (which would of itself be an equally correct rule) prevents the introduction of a cyclic rewrite strategy which would keep the system from terminating (*i.e.*, it preserves monotonicity).

We also note that the *mrouter* node will always be permitted to drop packets because of internal congestion; thus, another rule will always be applicable:

$$R_4 \quad : \quad x\ \textit{mrouter drop}\ y \ \triangleright\ \{x\ \textit{mrouter}\ y\}$$

Ultimately, we would like to be able to say something about the *mrouter* in any network arrangement. Assuming that all the relevant characteristics of intervening networks can be represented using models of *record* and *drop*, we can then define $\mathcal{A}$ as $\mathcal{SP}^*$ *mrouter* $\mathcal{C}$, where $\mathcal{S} = \{\textit{server}\}, \mathcal{P} = \{\textit{mrouter}, \textit{reord}, \textit{drop}\}$, and $\mathcal{C} = \{\textit{client}\}$. This definition handles two simplifications of the problem space for us up front: it excludes all arrangements which do not include at least one *mrouter* (*i.e.*, all arrangements in which we are not interested) and it removes all *reord* or *drop* which do not precede an *mrouter* (because they have no bearing upon the correctness of an *mrouter*). Given the already-stated four reductions, we can derive $\mathcal{A}_4$ (pictured in Figure 3) using the corresponding $f_1, \ldots, f_4$. $\mathcal{A}_4$ clearly still represents an infinite set of arrangements, so our methodology will require additional reductions in order to produce useful results.

[2]The second and third properties taken together represent the *closure* of the set described by these expressions under all reductions in $\mathcal{R}$.

[3]MPEG streams are structured as follows: each I frame signifies the beginning of a new group of pictures (GOP). Within a GOP, each P frame can only be decoded if the initial I frame and all previous P frames have been received. Similarly, a B frame can only be decoded if the previous P frame could be decoded and if all B frames between that P frame and itself have been received.

[4]We henceforth use "router" to mean a node in an (overlay) network that handles the forwarding of the packet to one (or more) other nodes in the network.

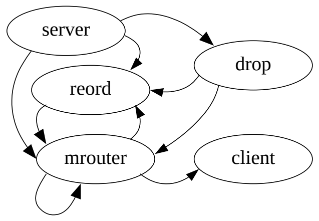

Figure 3. $\mathcal{A}_4$ **for basic MPEG-routing network**

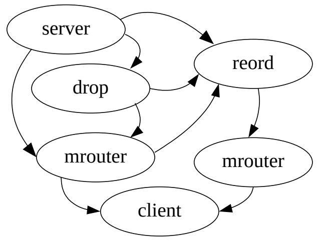

Figure 4. Acyclic automaton of $\mathcal{A}_6$ **for an ideal MPEG router**

Reducibility as Specification: Rather than thinking in terms of the reductions which a particular fully-specified application induces, one may prefer to state a design goal for an application in terms of a set of reductions which that application must satisfy. For example, we could state as a design property that a sequence of identical agents will be behaviorally equivalent to a single such agent; any protocol or implementation which disagrees with this property will fail to meet the design constraints.

We can then state, as an engineering goal, that *mrouter* should provably satisfy a set of reductions which yield a finite $\mathcal{A}_n$. The choice of proof method is not particularly important; whichever is best suited to the design and development environment can equally well be used. The following reductions would be sufficient, and make for an illustrative example of target reductions which could be set as correctness criteria for some *mrouter* formulation:

$$\begin{aligned} R_5 &: \quad x \textit{ mrouter mrouter } y \;\rhd\; \{x \textit{ mrouter } y\} \\ R_6 &: \quad x \textit{ reord mrouter reord mrouter } y \;\rhd \\ &\qquad \{x \textit{ drop reord mrouter } y\} \end{aligned}$$

If all six reductions are valid, then $\mathcal{A}_6$ has six members; thus, by testing only those six, we establish the behavior of *mrouter* in all possible network configurations. $\mathcal{A}_6$ is expressed by the automaton in Figure 4.

4.2.2 Confluence

Proof that this system of rewrite rules is confluent (and therefore gives rise to a canonical form) is straightforward.

Theorem 4.4. *The set of rewrite rules* $\mathcal{R} = \{R_1, \ldots, R_6\}$ *is confluent.*

Proof. We begin by proving termination. All rules but R_3 are length-decreasing (and therefore monotonic, so their corresponding $f^{(*)}$'s are defined); R_3 does not increase length and is clearly itself monotonic because it always moves *drop* nodes to the left and *reord* nodes to the right; the rules are clearly monotonic with respect to a composition of these two metrics (length as the major and *drop*/*reord* ordering as the minor component), thus $\mathcal{R}$ terminates. We can therefore establish its confluence using Newman's Lemma by demonstrating local confluence.

All pairings among R_1, R_2, R_4, R_5, and R_6 are clearly commutative over any chains in which their effects overlap. R_3 is similarly commutative with R_4, R_5, and R_6. This leaves only the pairings of R_3 with R_1 and R_2.

Consider the chain *reord reord drop*. By R_1 it is rewritten to *reord drop*; by R_3 it is rewritten to *reord drop reord*. Notice that these are indeed confluent: the former can then be rewritten by R_3 to *drop reord* and the later can be rewritten by R_3 (again) to *drop reord reord*, which can be rewritten by R_1 to *drop reord*. A similar case arises when R_3 is paired with R_2. Therefore, the lack of commutativity within these pairs is resolved by the existence of a succeeding rewrite strategy which brings their results into agreement.

Since $\mathcal{R}$ is both locally confluent and terminating, it is therefore confluent. □

It is easy to devise variations upon this $\mathcal{R}$ which express the same properties of the problem space but are difficult to make confluent; *e.g.*, an additional rule (based upon the rationale of R_4) which removes a *drop* preceding an *mrouter* is non-commutative with R_3, and converting this conflict into an additional rewrite rule does not result in confluence; many iterations of the Knuth-Bendix procedure are required to resolve this divergence. As another example, if R_3 is replaced with its inverse, a similar conflict arises between it and R_6, and the simplest apparent solution (resolved by including both R_3 and its inverse) does not work because the pair would support a non-terminating rewrite strategy.

4.2.3 Algorithms Resilient to Network Anomalies

With the constraints of R_5 and R_6 in mind, we have devised two variants of the algorithm in [13], one resilient to lost packets, the other resilient to reordered packets (with short reorder spans). Both require an additional packet header to make explicit the relationships among particular packet numbers; the drop-resilient algorithm is still constant time, and the reordering-resilient algorithm requires space and time linear with the degree of resilience (maximum span) desired. The algorithms themselves, along with sketched proofs of their agreement with R_5 and R_6, are omitted for want of space, and can be found in [7].

4.3. Web Intra-Cache Consistency

There is nothing in the CHAIN methodology which is intrinsically linked with finite-state model checking; any methodology which can give rise to proofs and which allows for the discovery of reduction/equivalence relations among sets of configurations can just as well act as the basis for

$$((\mathcal{S}\ (\textit{proxy-scrubber} \mid \textit{proxy-plain})^*) \mid$$
$$(\textit{server-btc}\ \mathcal{P}^*_{\textbf{clean}}\ \textit{cache-btc}\ (\textit{proxy-plain} \mid \textit{proxy-scrubber})^*) \mid$$
$$(\textit{server-btc}\ \mathcal{P}^*_{\textbf{clean}}\ \textit{cache-btcpush}\ \mathcal{P}^*)\ )\ \textit{client}$$

Where $\mathcal{P}_{\textbf{clean}} = \{\textit{proxy-plain}, \textit{cache-plain}, \textit{cache-btc}, \textit{cache-btcpush}\}$

Figure 5. Consistent Cache Arrangements for Theorem 4.5

defining our property of interest π and the set of reduction rules $\mathcal{R}$. As an example, in this section we show the application of this methodology to the characterization of a web cache system which employs the Basis Token Consistency protocol [4], a protocol whose correctness follows directly from the definition of vector clocks [11, 21] (its underlying conceptual mechanism).

For BTC, the interesting π is whether the client at the end of some arrangement $a \in \mathcal{A} = \mathcal{S}\mathcal{P}^*\mathcal{C}$ will be guaranteed to see a consistent sequence of responses, *i.e.*, one which is temporally non-decreasing (but not necessarily *recent*). If $\pi(a) = \textbf{true}$, then arrangement a will always cause the client's view of the server to be consistent (temporally non-decreasing); $\pi(a) = \textbf{false}$ indicates that arrangement a *can* provide a client with an inconsistent response. Basis Token Consistency (BTC) guarantees such consistency for any supporting cache downstream of a supporting server, regardless of the presence of intermediary inconsistent caches (so long as intermediary proxies do not repress response headers which they do not understand). This fundamental property of BTC gives rise directly to a rewrite rule which preserves π: any number of proxies which do not "scrub" headers (*i.e.*, proxies which do not flagrantly violate the HTTP specification) between a BTC server and a BTC downstream agent (client or cache) will not affect π and can therefore be rewritten out of the set of characteristic arrangements.

A simplified model of the web for BTC's purposes uses the following agents:

$$\begin{aligned}\mathcal{S} =& \{\textit{server-btc},\ \textit{server-plain}\},\\ \mathcal{P} =& \{\textit{proxy-scrubber},\ \textit{proxy-plain},\ \textit{cache-plain},\\ & \quad \textit{cache-btc},\ \textit{cache-btcpush}\},\\ \mathcal{C} =& \{\textit{client}\}.\end{aligned}$$

where *cache-btcpush* uses the end-to-end strong consistency extension [3]. The inclusion of $\mathcal{C}$ is pure sugar; the interesting property as far as π is concerned is the state of the furthest downstream cache, *i.e.*, the caching agent appearing closest to the end of the arrangement. Other types of agents besides the ones described can be modeled as particular sequences of these basic elements.

The definitions of standard proxying and proxy-caching in light of BTC's notion of consistency give rise to some basic reductions, such as the insertion of *proxy-plain* (cacheless proxy) agents having no effect, or indifference to the ordering of *proxy-scrubber* and *cache-plain* agents. These are reflected as reductions R_1 through R_6 in an Appendix of [7][5].. The definition and the correctness of BTC itself gives rise directly to 14 additional reductions, R_7 through R_{20} also found in [7].

These twenty rules, through the application of the CHAIN methodology, identify a homomorphic image $\mathcal{A}_{20}$ containing four member arrangements, described by the expression:

$$\mathcal{A}_\top = (\textit{server-plain}|\textit{server-btc})\ \textit{cache-plain}^{\leq 1}\ \textit{client}$$

where the two arrangements without a *cache-plain* are consistency-safe and the two containing a *cache-plain* are consistency-unsafe.

Confluence These twenty rules are not confluent because some members of $\mathcal{A}_{\textbf{false}}$ can be rewritten to both of the false members of $\mathcal{A}_{20}$. $\mathcal{A}_{\textbf{true}}$ has a similar problem. The system can easily be made confluent, however, by adding a set of finalizing rewrites which collapse the two "true" members of $\mathcal{A}_{20}$ into one (*server-btc client* $\triangleright$ *server-plain client*) and likewise for the two "false" members (*server-btc cache-plain client* $\triangleright$ *server-plain cache-plain client*). The new system of twenty-two rewrite rules is confluent and produces an $\mathcal{A}_\top$ with only two member arrangements; since π is a binary property, this implies that result of the rewriting process itself maps trivially to the value of $\pi(a)$ for any $a \in \mathcal{A}$.

Intuitively, we know that a caching system will provide the client with a consistent view under any of three circumstances: (1) there are no caches between the server (whether plain or BTC) and the client; (2) the server supports BTC, the last cache before the client is reached is a BTC cache, and there are no scrubbers between the BTC server and that final cache; (3) the server supports BTC, the system includes a *btcpush* cache, and there are no scrubbers between the server and a *btcpush* cache.

Theorem 4.5. *All caching arrangements which provide a client with a consistent view of server state (that is, all members of $\mathcal{A}_{\textbf{true}}$) will match the pattern stated in Figure 5.*

Proof. Similar to Theorem 4.3. A "safety pattern" is simply the compliment of a "failure pattern", so its validity is established by the same properties: first, whether it correctly partitions $\mathcal{A}_\top$; second, whether it defines a set which is closed under all reductions (that is, reductions preserve both membership and non-membership).

These properties are proven in an Appendix of [7]. □

5. Conclusion

In this paper we have presented CHAIN, an algebraic approach that enables the reduction of arbitrary arrangements of network protocol agents to a canonical, behaviorally-equivalent (with respect to some correctness property) representation, which is amenable to mechanical verification. Our methodology relies upon the discovery of a sufficient

[5]The reductions are excluded from this paper in the interest of brevity.

set of reduction/rewrite relationships, which establish homomorphism between subsets of the set of arrangements. We have applied our approach to the verification of safety properties of three examples: The HTTP request continuation protocol, an MPEG packet forwarding protocol for overlay networks, and the BTC web intra-cache consistency protocol.

Broader Vision and Research Agenda To a great extent, the programming of distributed applications over the Internet suffers from the same lack of organizing principles as did programming of stand-alone computers some thirty years ago. Primeval programming languages were expressive but unwieldy; software engineering technology improved not only through better understanding of useful abstractions, but also by automating the process of verification of safety properties both at compile time (e.g., type checking) and run time (e.g., memory bound checks). We believe that the same kinds of improvements can find their way into the programming of distributed Internet services. CHAIN is an instance of our broader goal of applying more rigorous disciplines to the specification and creation of networked protocols, programs, and services. The development of CHAIN is an important milestone for *ı*Bench, our ongoing initiative seeking to provide a sound framework for integrating a wide range of proof and verification strategies with the principles of design, development, compilation and execution of disciplined and safe programmable systems. The *ı*Bench Initiative's web pages can be found at `http://www.cs.bu.edu/groups/ibench/`.

References

[1] F. Baader and T. Nipkow. *Term* Re*writing and* All That. Cambridge University Press, 1998.

[2] J. Baeten and W. Weijland. *Process Algebra*. Cambridge University Press, 1990.

[3] A. D. Bradley and A. Bestavros. Basis token consistency: Extending and evaluating a novel web consistency algorithm. In *Workshop on Caching, Coherence, and Consistency (WC3)*, New York, June 2002. IEEE.

[4] A. D. Bradley and A. Bestavros. Basis token consistency: Supporting strong web cache consistency. In *Global Internet Worshop*, Taipei, Nov. 2002. IEEE.

[5] A. D. Bradley, A. Bestavros, and A. J. Kfoury. Safe composition of web communication protocols for extensible edge services. In *Workshop on Web Caching and Content Delivery (WCW)*, Boulder, CO, Aug. 2002.

[6] A. D. Bradley, A. Bestavros, and A. J. Kfoury. Safe composition of web communication protocols for extensible edge services. Technical Report BUCS-TR-2002-017, Boston University Computer Science, 2002.

[7] A. D. Bradley, A. Bestavros, and A. J. Kfoury. Systematic verifi cation of safety properties of arbitrary network protocol compositions using CHAIN. Technical Report BUCS-TR-2003-012, Boston University Computer Science, 2003.

[8] S. Chaki, S. K. Rajamani, and J. Rehof. Types as models: Model checking message-passing programs. In *POPL 2002*, Portland, OR, Jan. 2002.

[9] M. Chechik, B. Devereux, and A. Gurfi nkel. Model-checking infi nite state-space systems with fi ne-grained abstractions using SPIN. In *8th SPIN Workshop*, Toronto, 2001.

[10] J. Dingel and T. Filkorn. Model checking for infi nite state systems using data abstraction, assumption-commitment style reasoning and theorem proving. In P. Wolper, editor, *Proceedings of the 7th International Conference on Computer Aided Verifi cation*, volume 939, pages 54–69, Liege, Belgium, 1995. Springer Verlag.

[11] C. Fidge. Logical time in distributed computing systems. *Computer*, 24(8):28–33, Aug. 1991.

[12] T. G. Griffi n and G. Wilfong. On the correctness of IBGP confi guration. In *ACM SIGCOMM*, Pittsburgh, PA, Aug. 2002. ACM.

[13] D. He, G. Muller, and J. L. Lawall. Distributing MPEG movies over the internet using programmable networks. In *International Conference on Distributed Computing Systems (ICDCS)*, July 2002.

[14] G. J. Holzmann. Designing bug-free protocols with SPIN. *Computer Communications Journal*, pages 97–105, Mar. 1997.

[15] D. Jackson. Abstract model checking of infi nite specifi cations. In *Proceedings of Formal Methods Europe*, Barcelona, Oct. 1994.

[16] D. E. Knuth and P. Bendix. Simple word problems in universal algebra. *Computational Problems in Abstract Algebra*, pages 263–297, 1970.

[17] B. Krishnamurthy, J. C. Mogul, and D. M. Kristol. Key differences between HTTP/1.0 and HTTP/1.1. In *Proceedings of the WWW-8 Conference*, Toronto, May 1999.

[18] A. Kučera and P. Jančar. Equivalence-checking with infi nite-state systems: Techniques and results. In *Proceedings of 29th Seminar on Current Trends in Theory and Practice of Informatics (SOFSEM 2002)*, pages 41–73. Springer-Verlag, 2002.

[19] M. Leuschel and T. Massart. Infi nite state model checking by abstract interpretation and program specialisation. In A. Bossi, editor, *Logic-Based Program Synthesis and Transformation (LOPSTR'99), LNCS 1817*, pages 63–82, Venice, Italy, 2000.

[20] N. Lynch and F. Vaandrager. Forward and backward simulations – part I: Untimed systems. *Information and Computation*, 121(2):214–233, Sept. 1995.

[21] F. Mattern. Virtual time and global states of distributed systems. In *Proc. Parallel and Distributed Algorithms Conf.*, pages 215–226, 1988.

[22] J. Mogul. Is 100-Continue hop-by-hop?, July 7, 1997. HTTP-WG Mailing List Archive, http://www-old.ics.uci.edu /pub/ietf/http/hypermail/1997q3/.

[23] V. Paxson. End-to-end internet packet dynamics. *IEEE/ACM Transactions on Networking*, 7(3):277–292, 1999.

[24] S. Shenker. Where's the science? Keynote Address presented at the NSF ANIR Principle Investigators Meeting, Reston, VA, Jan. 2003.

[25] J. Touch, D. Wagner, and J. Walrand. Panel recommendations on "Network Trustworthiness". Presented at the NSF ANIR Principle Investigator Meeting, Reston, VA, Jan. 2003.