Information Processing Letters 115 (2015) 694–698

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

Non-deterministic transducer models of retransmission protocols over noisy channels

Jay Thakkar, Aditya Kanade *

Indian Institute of Science, Bangalore, India

ARTICLE INFO

Article history:
Received 21 August 2014
Received in revised form 6 January 2015
Accepted 17 April 2015
Available online 21 April 2015
Communicated by J.L. Fiadeiro

Keywords:
Formal methods
Protocols
Noisy channels
String transducers

ABSTRACT

Retransmission protocols such as HDLC and TCP are designed to ensure reliable communication over noisy channels (i.e., channels that can corrupt messages). Thakkar et al. [15] have recently presented an algorithmic verification technique for *deterministic* streaming string transducer (DSST) models of such protocols. The verification problem is posed as equivalence checking between the specification and protocol DSSTs. In this paper, we argue that more general models need to be obtained using *non-deterministic* streaming string transducers (NSSTs). However, equivalence checking is undecidable for NSSTs. We present two classes where the models belong to a sub-class of NSSTs for which it is decidable.

1. Introduction

Retransmission protocols use cyclic redundancy check and sliding window protocols for error detection and control respectively [14]. TinyOS serial communication protocol, Philips bounded retransmission protocol (BRP), high-level data link control (HDLC), and transmission control protocol (TCP) are examples of widely used retransmission protocols that provide reliable communication over *noisy channels*, i.e., channels that can corrupt messages.

1.1. Motivation for transducer based modeling of retransmission protocols

In a recent work, Thakkar et al. [15] present an approach of transducer based modeling of retransmission protocols over noisy channels. They show that the usual approach of abstracting message contents by symbolic constants (e.g. [11]) is inadequate in this setting. In particular, they illustrate that unless the message contents are modeled as *bit strings*, the noisy channel can give rise to a sequence of message corruptions inducing the receiver to deliver an incorrect sequence of messages to its client (see [15], Section 2). Thus, we need an expressive framework to precisely model retransmission protocols. Modeling them as finite-state machines communicating asynchronously over unbounded FIFO channels however does not yield a decidable framework [6].

They show that *deterministic streaming string transducers* or *DSSTs* [2,3] provide an intuitive and expressive modeling framework of retransmission protocols. DSSTs can model different classes of retransmission protocols such as those based on stop-and-wait, go-back-n and selective-repeat sliding window protocols [14]. In these models, the length of a message string and the number of retransmission rounds can be *unbounded*. Even in the presence of these sources of unboundedness, the protocol verification problem — formalized as equivalence checking problem over DSSTs — is *decidable*.

1.2. Deterministic protocol models

We first review the approach in [15]. In this approach, the protocol components — sender and receiver — and the specification are modeled using deterministic string

* Corresponding author.
E-mail address: kanade@csa.iisc.ernet.in (A. Kanade).

http://dx.doi.org/10.1016/j.ipl.2015.04.009
0020-0190/

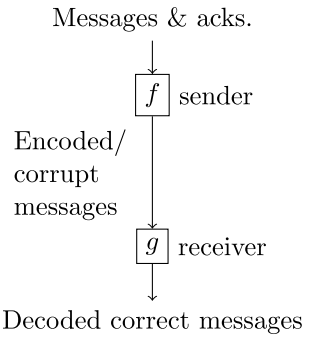

Fig. 1. Sender and receiver transducer models.

transducers. As shown in Fig. 1, the input to a *sender transducer* f is a sequence of strings representing messages to be transmitted as well as acknowledgements sent by the receiver where both messages and acknowledgements are bit strings. The sender's output is the sequence of encoded or corrupt messages that arrive at the receiver *over the noisy channel* across all rounds of transmission. That is, the noisy behavior of the channel and the protocol's retransmission logic are modeled in the output of the sender. The *receiver transducer* g (1) recognizes and discards corrupt messages, and (2) extracts and outputs decoded values of correctly received messages. The *protocol transducer* $p \equiv g \circ f$ is obtained by sequential composition of the sender and receiver transducers where $(g \circ f)(x) = g(f(x))$.

The *specification transducer* h captures the desired end-to-end behavior of the protocol. It requires that (1) the messages be delivered by the receiver to its client in the same order in which they were received by the sender from its client and (2) the protocol delivers exactly those messages that are positively acknowledged (not corrupted by the channel). The verification problem is posed as *functional equivalence* between the transducers h and p, that is, whether $dom(h) = dom(p)$ and for all $w \in dom(h)$, $h(w) = p(w)$. Here, the output of the transducer g is the input to the client of the receiver. Another transducer g' can be constructed, in a similar manner, to model the acknowledgements that the receiver would generate for the sender component and verified separately.

1.3. Limitation of deterministic models

The deterministic models presented in [15] use a *fixed* string *ERR* in the sender's output to capture the noisy behavior of the channel. As an example, suppose the input to the sender transducer is $M\sharp a$, where $\sharp$ is the end-marker of the message string M and a is an acknowledgement. If $a = 0$ then it is a negative acknowledgement indicating that the previously transmitted message was received incorrectly. Otherwise, it indicates correct reception. Following the approach of [15], let the sender transducer f be: $f(M\sharp 1) = M$ and $f(M\sharp 0) = ERR$.[1] The specification transducer is $h(M\sharp 1) = M$ and $h(M\sharp 0) = \epsilon$ where ϵ is the empty string. A receiver transducer $g(ERR) = \epsilon$ and $g(M) = M$ when $M \neq ERR$ is *verified* to be correct as h is equivalent to $g \circ f$.

Now, consider a *non-deterministic* sender transducer f' such that $f'(M\sharp 1) = M$ and $f'(M\sharp 0) = M'$ where M' is an arbitrary corrupt form of M. With this, the protocol model $g \circ f'$ would deliver a *corrupt* message $M' \neq ERR$ to the receiver's client. In other words, with the deterministic sender f, we cannot ascertain (1) whether the receiver g handles all forms of corrupt messages and (2) whether the protocol delivers the messages correctly in the presence of *arbitrary corruption*. To establish these properties, we need a *non-deterministic* model of the sender transducer (similar to f') that emits arbitrarily corrupt messages instead of a fixed error string.

1.4. Our approach

In this work, we propose to use *non-deterministic streaming string transducers* or *NSSTs* [4] for modeling the sender. NSSTs are closed under sequential composition (required to compute the protocol transducer) but equivalence checking for NSSTs is undecidable. For a sub-class of NSSTs, called *functional NSSTs* [4], equivalence checking is PSPACE-COMPLETE. We observe that the receiver transducer is *deterministic* due to the protocol semantics. Unfortunately, the sequential composition of an NSST and a DSST does *not* necessarily yield a functional NSST (see Section 4 for an example).

Nevertheless, we show that for the following two interesting classes of protocols, the senders and receivers are of the form that when composed, they result in functional NSSTs:

1. *Bounded retransmission rounds per message*: Here, the messages can be corrupted arbitrarily but each message can be retransmitted up to a certain *fixed* number of rounds. This class is motivated by Philips bounded retransmission protocol (BRP) [10].
2. *Bounded non-determinism in message corruption*: The messages can be corrupted non-deterministically but only in *finitely* many ways, whereas, the messages can be retransmitted an unbounded number of times.

In both these cases, the message lengths and the total number of messages transmitted are *unbounded*. We show that both these classes result in functional-NSST protocol models and hence, can be verified *algorithmically*.

In practice, the implementations of a protocol may differ in the choice of the number of retransmission rounds or the error detection mechanism. The protocol models we construct in this paper are generic, and different concrete models can be obtained by instantiating them and verified individually.

The expressive power of functional NSSTs is same as that of DSSTs [4]. As a consequence, our results also imply that the above classes of protocols can be modeled directly as DSSTs. However, our *modular* approach of modeling the (non-deterministic) sender and the (deterministic) receiver

[1] In the actual models (and real protocols), a message is encoded with a checksum and a sequence number. We postpone these details to the latter part of the paper.

separately is much simpler compared to modeling the end-to-end protocol directly.

2. Background

A *non-deterministic* streaming string transducer (NSST) is described by a tuple $(Q, \Sigma_1, \Sigma_2, X, E, F, q_0)$, where Q is a finite set of states, Σ_1 and Σ_2 are finite input and output alphabets respectively, X is a finite set of *string variables*, E is a set of transitions which is a finite subset of $(Q \times \Sigma_1 \times A \times Q)$ where A is a set of *copyless assignments* from X to $(X \cup \Sigma_2)^*$ such that for each $x \in X$ and $\alpha \in A$, x appears at most once in $\{\alpha(y) \mid y \in X\}$, F is a partial output function from Q to $(X \cup \Sigma_2)^*$ such that for each $q \in Q$, $F(q)$, if defined, contains an $x \in X$ at most once, and $q_0 \in Q$ is the initial state.

An NSST f reads an input string w in a single left-to-right pass and follows possibly non-deterministic transitions based on the input symbol. Initially, all the string variables X are initialized to empty strings. During each transition, the NSST updates values of all variables *simultaneously* according to the assignment α labeling the transition. After reading the input string w completely, suppose f reaches a set of states $Q_w \subseteq Q$. Let $q \in Q_w$ and Λ_q be the set of mappings of type $X \to \Sigma_2^*$ obtained by repeated assignments to X while reaching q. An output of f on w at q is $\alpha_q(F(q))$ obtained by replacing each string variable $x \in X$ in $F(q)$ by $\alpha_q(x)$ for an $\alpha_q \in \Lambda_q$. If $F(q)$ is undefined then $\alpha_q(F(q))$ is also undefined. The output of f on w, denoted by $f(w)$, is the set of strings $\{\alpha_q(F(q)) \mid q \in Q_w, \alpha_q \in \Lambda_q$ and F is defined for $q\}$.

An NSST f thus defines a relation between input and output strings. If for each $w \in \Sigma_1^*$, $f(w)$ is either an empty or a singleton set then the NSST is called a *functional NSST*. In a *deterministic* streaming string transducer (DSST), the transition relation E contains at most one transition for every state $q \in Q$ and $a \in \Sigma_1$. Clearly, $f(w)$ for any w is either an empty or a singleton set for a DSST f. The expressive power of functional NSSTs and DSSTs is identical [4]. The difference between a functional NSST and a DSST is that a DSST follows a unique transition sequence for any input w, whereas, a functional NSST may follow multiple sequences but all resulting in the same output (if defined).

The *sequential composition* $g \circ f$ is defined as: $(g \circ f)(w) = \{w' \mid \exists w'' \in f(w)$ and $w' \in g(w'')\}$. Two functional NSSTs f and g, both $\Sigma_1^* \to \Sigma_2^*$, are *(functionally) equivalent* if $dom(f) = dom(g)$ and $\forall w \in dom(f) : f(w) = g(w)$. We state some known results [2–4]: (1) DSSTs, NSSTs, and functional NSSTs are closed under sequential composition. (2) Equivalence checking is decidable for DSSTs and functional NSSTs, but undecidable for NSSTs.

3. Non-deterministic protocol models

We first informally describe the retransmission protocols. In a retransmission protocol, the sender receives messages to be transmitted from its client and message acknowledgements from the receiver. The sender prepends a unique *sequence number* (from a finite interval) to a message and maintains pending messages in a set of buffers. *Pending messages* are those which are transmitted by the sender but have not been positively acknowledged by the receiver so far. The set of sequence numbers associated with pending messages is called a *sliding window*. A message is encoded with a *checksum* before it is transmitted. The receiver checks whether the checksum of a received message is correct. If it is, it sends a *positive acknowledgement* to the sender else it sends a *negative acknowledgement* — indicating that the message was corrupted by the noisy channel. The receiver extracts the correct messages by removing the checksum and sequence number and passes them to its client; and drops corrupt messages. If a message is positively acknowledged, the sender replaces it with the next message from its client. It retransmits a message until the message is positively acknowledged, ensuring reliable communication over a noisy channel.

We use the non-deterministic and deterministic versions of streaming string transducers for modeling the sender and receiver components. They provide a natural way of modeling these protocols, e.g., the buffers to hold pending messages can be modeled directly by string variables. We now explain two classes of protocols: protocols with bounded retransmission rounds per message and protocols with bounded non-determinism in message corruption.

3.1. Protocol class I: bounded retransmission rounds per message

The first class is motivated by the Philips bounded retransmission protocol (BRP) [10]. For the models in this class, the message lengths and the number of messages transmitted are *unbounded*. Further, the messages can be corrupted arbitrarily. However, the number of times an individual message can be retransmitted is *bounded*, similar to BRP. For simplicity, we explain the modeling with the stop-and-wait protocol [14]. Similar models can be derived for go-back-n and selective repeat sliding window strategies (see [14]).

Fig. 2 shows the abstract model of the sender based on the stop-and-wait protocol. It uses only one buffer and a one-bit sequence number. In other words, at most one message can be pending at any point of time. In this model, the input to the *sender transducer* f is a sequence of strings denoted by M, *ack* and *nack* where $M \in \{0,1\}^*$ is a message bit string, and $ack \in \{0,1\}^k$ and $nack \in \{0,1\}^k$ are respectively fixed-length strings representing positive and negative acknowledgements. In the actual models, these string literals are expanded into bit strings. For a message M and a sequence number i, let $enc(i, M)$ append the checksum c to $i.M$. The checksum can be tracked in the states of the transducer and $enc(i, M)$ can be modeled through suitable transitions. Suppose the checksum can detect up to k bit flips. Then, $crupt(i, M)$ can non-deterministically flip up to k bits of $enc(i, M)$. Any arbitrary corruption of up to k bits can be modeled in the transducer.

Each state of the abstract model has three components: component #1 indicates whether the buffer is empty (E)

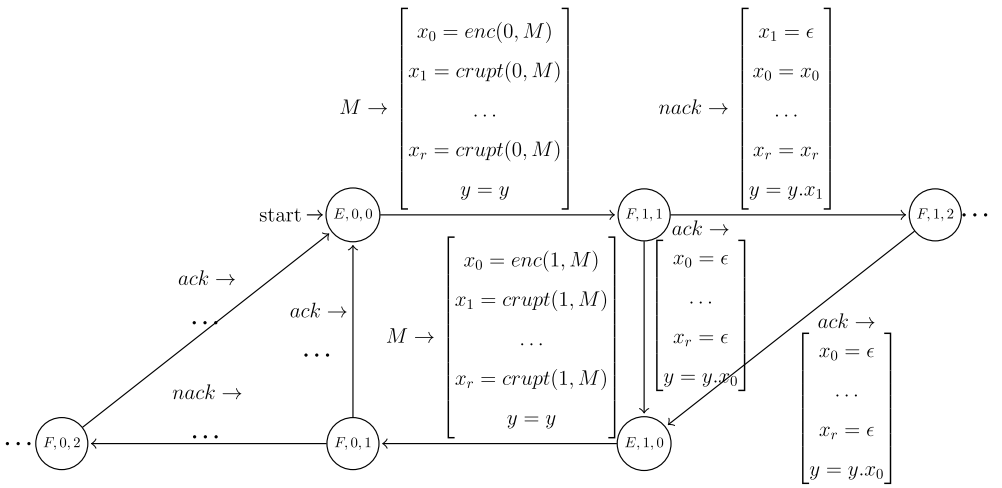

Fig. 2. Abstract model of the sender transducer based on the stop-and-wait protocol.

or full (F), component #2 represents the sequence number to be used for the next incoming message M, and component #3 tracks the round of retransmission for the current message. As shown in Fig. 2, the start state is $(E, 0, 0)$. The number of retransmission rounds per message is *bound* by a constant, say r. The transducer uses $r + 2$ string variables $x_0, x_1, \ldots, x_r, y$. String variable y accumulates the messages that will be delivered to the receiver across all transmission rounds, and x_0 is used to store a correct encoding of the input message and corresponds to the buffer used by the protocol. String variables $x_1, \ldots, x_r$ are used to store *non-deterministically generated* corrupt messages for rounds 1 to r respectively.

As shown in Fig. 2, in a state $(E, i, 0)$, if the sender is handed a message M by its client then it stores $enc(i, M)$ in x_0, simultaneously generates r corrupt copies using $crupt(i, M)$ and stores them in $x_1, \ldots, x_r$, but does not modify y. It moves to a state $(F, i + 1 \ mod \ 2, 1)$. If *nack* arrives in a state (F, i, j) then y is updated to $y.x_j$ (reflecting the fact that the message arrived in a corrupt form at the receiver in the jth round) and x_j is set to ϵ while keeping other variables unchanged. The transducer moves to a state $(F, i, j + 1)$. For example, see the transition from $(F, 1, 1)$ to $(F, 1, 2)$ in Fig. 2. On *ack*, y is updated to $y.x_0$, and $x_0, \ldots, x_r$ are set to ϵ, while moving to a state $(E, i, 0)$. For example, see the transition from $(F, 1, 1)$ to $(E, 1, 0)$ in Fig. 2. At (F, i, r), only a positive acknowledgement is permitted. The output function maps each state to y.

Receiver Transducer The output of the sender transducer is the input to the *receiver transducer* g. The receiver reads an encoded message and extracts the sequence number i and copies the message M into a string variable u. The finite control tracks the checksum of $i.M$. If the checksum accompanying the received message matches the computed checksum then u is appended to a string variable v which accumulates output of the receiver. Otherwise, the input message is corrupt and is discarded by not modifying v and setting u to ϵ. The output function maps each state to v.

Specification Transducer The input to the *specification transducer* h is same as that of the sender and its output is the messages delivered to the receiver's client. It uses two string variables: s to store the current message and t to accumulate the output. It looks ahead at the acknowledgements of a message and appends s to t only when *ack* is seen for it. It does *not* involve message encoding through sequence number and checksum, or detection of corrupt messages.

3.2. Protocol class II: bounded non-determinism in message corruption

Along similar lines, we derive another class of models in which the number of retransmission rounds per message can be unbounded but the sender non-deterministically selects a corrupt message from a finite set $\{ERR_1, \ldots, ERR_n\}$. Since these are hard-coded strings, it does not require string variables to maintain the corrupt copies separately for each round unlike Fig. 2. Thus, only two variables, analogous to x_0 and y, are required.

3.3. Extensions

A string transduction t is called a *linear-size increase transduction* if there exists a constant k such that for every input w and an output $w' \in t(w)$, the length of w' is at most k times the length of w [13]. NSSTs model such transductions [4]. It can be seen that the sender transducers proposed above define linear-size increase transductions. However, if we (1) make the number of retransmission rounds unbounded in case I or (2) permit unbounded non-determinism (use of $crupt(i, M)$) in case II, the resulting transducers are no longer linear-size increase transducers and cannot be modeled as NSSTs.

4. Algorithmic verification

The protocol verification problem is formalized as equivalence checking between the protocol and specification transducers. In order to obtain the protocol transducer,

we sequentially compose the sender NSST and the receiver DSST. The specification transducer is a DSST. For algorithmic verification, we require the protocol transducer to be a functional NSST. In general, the sequential composition of an NSST and a DSST is not necessarily a functional NSST. For example, let $f'(a) \in \{b, c\}$ and $g'(b) = d$ and $g'(c) = e$. Here, f' is an NSST and g' is a DSST. However, $g' \circ f'$ maps the input a to d or e non-deterministically. We now show that the sequential composition of our sender and receiver models does result in functional NSSTs.

The sender being an NSST, f is a one-to-many function. The sender may output a non-deterministically corrupted message M' (instead of the original message M). A correct receiver must drop all corrupt messages. In our construction, the variable u is set to ϵ and the output variable v is not modified for a corrupt message (see Section 3.1). Thus, the receiver g is a many-to-one function, mapping all corrupt messages to ϵ. The sender has non-determinism only in the choice of bit flips but all bit flips are detected by the receiver. Thus, for all $w \in dom(f)$, $(g \circ f)(w)$ is defined *uniquely* but the composed transducer may follow different transition sequences for different choices of bit flips.

Theorem 1. *In our protocol models, the sequential composition of a non-deterministic sender and a deterministic receiver results in a functional NSST.*

Our framework can be used by developers to model other retransmission protocols. If the receiver modeled by a developer fails to identify some corrupt messages then the protocol transducer would become non-functional. Fortunately, checking *functionality* of an NSST is in PSPACE [4]. Thus, we can automatically detect if the receiver is modeled incorrectly, causing non-determinism in the output of the protocol transducer. Only if $g \circ f$ passes the functionality check, we should check equivalence between the specification h and $g \circ f$.

5. Related work

The sender transduction f (as well as transducers p and h) performs *regular look-ahead* at the acknowledgements. Therefore, *single-pass* rational transducers cannot define f. The transduction f however is MSO-definable [8,9]. DSSTs are finite-state descriptions of MSO-definable string (MSOS) transductions. NSSTs are equivalent to non-deterministic MSOS transductions [4].

Our models handle unbounded message bit strings. The verification problem for finite-state machines communicating asynchronously over unbounded, perfect FIFO channels is undecidable [6]. We use transducers instead. For unbounded lossy FIFO channels, reachability and eventuality are decidable [1]. The semantics of lossy channels is orthogonal to the notion of noisy channels [15]. Several approaches for verification of sliding window protocols use model checking (see survey [5]) where the message contents are abstracted by symbolic constants or use semi-automated reasoning. While we model noisy channels through non-determinism in the sender's output, these have also been modeled through probabilistic semantics using formalisms of π-calculus [7] and value-passing CCS with noisy channels [12].

Acknowledgement

This work is funded partially by the Robert Bosch Center for Cyber Physical Systems at the Indian Institute of Science.

References

[1] P.A. Abdulla, B. Jonsson, Verifying programs with unreliable channels, Inf. Comput. 127 (2) (1996) 91–101.
[2] R. Alur, P. Cerný, Expressiveness of streaming string transducers, in: FSTTCS, 2010, pp. 1–12.
[3] R. Alur, P. Cerný, Streaming transducers for algorithmic verification of single-pass list-processing programs, in: POPL, 2011, pp. 599–610.
[4] R. Alur, J.V. Deshmukh, Nondeterministic streaming string transducers, in: ICALP, 2011, pp. 1–20.
[5] F. Babich, L. Deotto, Formal methods for specification and analysis of communication protocols, IEEE Commun. Surv. Tutor. 4 (1) (2002) 2–20.
[6] D. Brand, P. Zafiropulo, On communicating finite-state machines, J. ACM 30 (2) (1983) 323–342.
[7] Y. Cao, Reliability of mobile processes with noisy channels, IEEE Trans. Comput. 61 (9) (2012) 1217–1230.
[8] B. Courcelle, Graph operations, graph transformations and monadic second-order logic: a survey, Electron. Notes Theor. Comput. Sci. 51 (2001) 122–126.
[9] J. Engelfriet, H.J. Hoogeboom, MSO definable string transductions and two-way finite-state transducers, ACM Trans. Comput. Log. 2 (2) (2001) 216–254.
[10] J. Groote, J. Pol, A bounded retransmission protocol for large data packets, in: AMAST, 1996, pp. 536–550.
[11] G.J. Holzmann, The model checker SPIN, IEEE Trans. Softw. Eng. 23 (5) (1997) 279–295.
[12] S. Huang, Y. Cao, H. Wang, W. Qu, Value-passing CCS with noisy channels, Theor. Comput. Sci. 433 (2012) 43–59.
[13] M.P. Schützenberger, Sur les relations rationnelles entre monoides libres, Theor. Comput. Sci. 3 (2) (1976) 243–259.
[14] A. Tanenbaum, D. Wetherall, Computer Networks, Pearson, 2010.
[15] J. Thakkar, A. Kanade, R. Alur, Transducer-based algorithmic verification of retransmission protocols over noisy channels, in: FMOODS/FORTE, 2013, pp. 209–224.