On-line timed protocol trace analysis based on uncertain state descriptions[*]

Marek Musial
Technical University of Berlin
Department for Technical Computer Science, Real Time Systems Group

www: `http://www.cs.tu-berlin.de/~musial`
email: `musial@cs.tu-berlin.de`

Abstract
This paper presents a new approach to the task of passive protocol tracing. The method called *FollowSM* for the first time meets all requirements of practical in-field use, including the checking of time constraints, the independence of the current state when starting the analysis, the admittance of nondeterminism, and on-line real time analysis capability. This is achieved by a suitable modeling of the implementation under test and the generalization of the tracing algorithm to operate on state information with any degree of uncertainty. *FollowSM* has been implemented as a prototype system and proved capable of minimizing the time required for troubleshooting.

Keywords
Trace analysis, time constraints, EFSM model, real time, passive monitoring

1 INTRODUCTION

The area of the formal specification, verification, and testing of communication protocols and their implementations has been a fertile research field from the late seventies up to today. But with regard to the testing of protocol implementations against an underlying formal specification, researchers have ever been focusing their interest on *active testing* (see e.g. [vBP94] for an overview or [FvB91, ISO91, TKB91]). Active testing means that a special tester system *actively* confronts the implementation under test (IUT) with certain preselected input messages and awaits the IUT's reactions. These reactions are matched against expected message patterns to obtain some verdict about the implementation's conformance with the protocol specifica-

[*]This work has been made possible by a grant of Siemens AG, Germany, who have applied for a patent on the presented approach and are planning to implement it as a feature of their protocol analyzers.

tion. All input messages and output message patterns are typically determined before test time, constituting *test cases* and *test suites*. Their derivation out of formal specifications is one of the key issues in the protocol engineering area (just a few examples: [A$^+$88, Bri88, LL91]).

On the other hand, relatively little work has been presented about *protocol trace analysis*. This concept means that the IUT is *passively* observed to obtain a *trace* of its observable interactions at some of its interaction points (IPs). The tester does not at all interfere with the communication progress. It has to answer the question whether there exists a sequence of internal actions permitted by the protocol specification that can accept the inputs in the observed trace, producing the observed outputs.

Apart from the well-known fact that testing can never prove the absence of faults but only sometimes their presence, testing by trace analysis provides a few specific advantages over active testing:

- It can be performed while the whole communication system is working under normal conditions. This means that it is highly probable that faults causing operational problems do occur during the test process as well.
- Sometimes PICS/PIXIT-parameters or user-settings might cause interworking problems between protocol implementations. These aspects cannot be covered by conformance tests of an isolated IUT.
- Passive testing within a largely functional communication system is possible without interrupting its operation.

This paper introduces a new non protocol-specific solution for passive trace analysis called *FollowSM* (for "follow a state machine"). To the author's knowledge, *FollowSM* is the first approach that addresses *all* practical requirements of the in-field trace analysis of modern protocols. These include the validation of timing constraints, the ability to start the analysis at any intermediate communication state, the admission of almost any kind of nondeterminism and efficient on-line operation. The method presented has been implemented in a prototype trace analyzer and applied to relevant B-ISDN protocols, i.e. Q.2110 (SSCOP) [IT94a] and Q.2931 [IT94b].

The remainder of this section lists related work from the literature. In section 2 the above-mentioned requirements on the presented method are explained in more detail. Section 3 gives a formal specification of the protocol model the approach is based on and defines the task of basic trace analysis. In section 4 trace analysis is extended to cover uncertain state information for the handling of initial states. Section 5 outlines how the formal model has been implemented in a prototype trace analyzer. In section 6 some experimental results are presented, and section 7 concludes the paper.

1.1 Related Work

Some of the first papers providing an overview of the trace analysis of communication protocols are [JvB83] and [UP86]. These papers primarily focus on the validation of refined specifications, but they already establish basic concepts and suggest solutions related to the field of the testing of protocol implementations.

An implementation of a passive monitor for the simultaneous trace analysis of several OSI layers is described in [CL91], with some emphasis put on a sample application to the X.25 protocol. This system is based on state machines coded in C and begins its analysis after a complete link reset.

The paper [KCV92] presents an approach which first enumerates *all* action paths possible due to the FSM projection of a restricted Estelle [ISO87a] specification, then deletes paths infeasible due to state variables and interaction parameters by symbolic evaluation. This method is very appealing from a theoretical point of view, especially since it may be used for both trace analysis and for test case generation. The fact that it operates on paths rather than on states might cause complexity problems with respect to on-line operation, particularly in the case of transitions that do not change the FSM state but are solely controlled by the data part of the protocol.

In [BvBDS91] trace analysis against LOTOS [ISO87b] specifications is investigated and the analysis tool *TETRA* is presented. Its Prolog implementation of the trace analysis algorithm utilizes uninstantiated Prolog variables to cope with nondeterministic choices and value generation. However, LOTOS (without extensions such as in [Sch94]) does not include time, and on-line evaluation tends to be inefficient.

The excellent work [EvB95] presents the *Tango* compiler, which translates a single-module Estelle specification without *delay* statements into a protocol-specific trace analyzer. This approach is elegant due to the direct use of Estelle, and all major issues of practical on-line trace analysis are discussed.

2 DESCRIPTION OF THE PROBLEM

This section describes the problem solved by the *FollowSM* approach to on-line timed protocol trace analysis in an informal way. The requirements stated below have been collected as a set of minimal demands for the in-field use of a trace analyzer as a troubleshooting tool.

Figure 1 depicts the context for the operation of the *FollowSM* trace analyzer. The latter gets all the PDUs exchanged between the IUT and its peer ("B") as its input, but does not interfere with communications in any way. In contrast to a protocol verification setting [Boc78], the analyzer shall not care about the validity of any inputs from B to the IUT, since it has to verify whether the IUT correctly reacts to possible protocol violations of its peer entity.

Since the approach presented here was designed for practical in-field use, there is no intention to observe additional interaction points (IPs), which would require special knowledge about the observed implementation and something like an upper tester (UT).

For practical application to modern protocols, *FollowSM* is required to observe and validate the PDU timing. Time checking in the context of trace analysis has to utilize tolerance intervals. For the apparent (observed) action timing will always deviate from the exact times specified because of signal propagation times, buffer delays and message transmission times.

Some complication arises from the fact that observations made on a bidirectional channel cannot be considered fully sequential. Figure 2 depicts the atomic case that

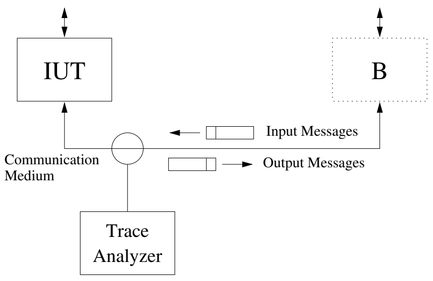

Figure 1 Operational context of *FollowSM*.

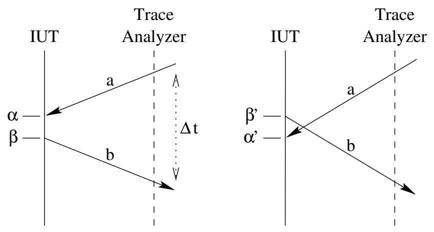

Figure 2 Ambiguous ordering of internal actions.

leads to nondeterministic action ordering: The trace analyzer first observes an input a to the IUT, and some time later an output b of the IUT. The two PDUs might, but need not, have crossed. This means both of the action sequences $\langle\alpha, \beta\rangle$ and $\langle\beta', \alpha'\rangle$ respectively could explain the observation. This is usually not an issue in active testing.

In [EvB95] this kind of problem is solved by an additional option to disable sequence checking among certain queues. In the timed approach presented here, it is necessary to be more restrictive: If and only if the time interval Δt falls below a specified upper bound, the "output first" action sequence has to be taken into consideration as well.

A trace analyzer should be able to start its analysis *at any intermediate state of the communication*, for it is practically undesirable to have to wait for some – possibly rare – synchronization conditions. The same ability is required after the detection of an error in the IUT's behavior, because in any such case the subsequent protocol state of the IUT is completely uncertain. *Ezust* and *Bochmann* [EvB95] suggest the attachment of an "undefined" attribute to all state variables in the analyzer's state machine as a possible enhancement to their approach. In this paper a more general approach is introduced which will turn out to cover the "undefined" attribute solution as a special case.

Another indispensable requirement to the *FollowSM* method is the processing of actually nondeterministic protocol specifications. Even if the target protocol is basically deterministic, uncertain initial states, ambiguous action ordering, and hidden requests at the upper service access point (SAP) will introduce nondeterminism.

In order to run a trace analyzer on-line, i.e. simultaneously with the communication, mainly the following three conditions have to be met: The analyzer's message processing speed must catch up with the message throughput of the communication, the algorithm has to be prepared for new messages to arrive during the analysis process, and its accesses of the trace data must be local. The last two conditions constitute qualitative design guidelines for the *FollowSM* algorithm, whereas the first one is just a quantitative condition.

3 PROTOCOL MODELING AND BASIC TRACE ANALYSIS

The analysis algorithm is based on a single extended finite state machine (EFSM) specification of the observed protocol. The employed EFSM model carries the necessary extensions to define timing restrictions and some specializations that guarantee the efficient computability of the trace analysis problem. This model will be referred to as *observable timed extended finite state machine*, OTEFSM, during the rest of the paper, and formally defined in this section. The restriction to a *single* state machine for protocol modeling imposes no mathematical limitation because *extended* finite state machines can simulate Turing machines by state variables with infinite range.

The approach presented here focuses on the computational aspects of practical trace analysis. How to derive the single OTEFSM specifying the permissible IUT behavior given a formal protocol specification in Estelle [ISO87a], LOTOS [ISO87b] or SDL [CCI87] is beyond the scope of this paper. The experimental results reported in section 6 are based on manual conversions of the respective SDL specifications. Since it is highly desirable to directly base the trace analysis on existing formal specifications, work is currently in progress to automate this transformation task for the language SDL.

3.1 Observable Timed Extended Finite State Machine

In the OTEFSM model, time is represented by natural numbers, but countability is not relied on in any context.

An OTEFSM M is a quadruple $M = (Q, T, \Sigma, \Delta, d)$ such that:

- Q is a set of *simple states*, denoting the part of the protocol machine's internal state information that is *not time-related*. Q may be infinite.
- T is a finite and possibly empty set of *timer labels*.
- Σ is a possibly infinite set of *input or output symbols*.
- Δ is a finite set of *transitions*. Their structure is refined below.
- $d \in \mathbb{N}$ is the maximum *output delay*, the amount of time that may pass before generated outputs become visible.

In the following, indexing by name will be used to clarify the component relation whenever necessary. For example, Q_M denotes the state space component of the OTEFSM M. Indexing by natural numbers will sometimes mean projection, e.g. $p_2 = 4$ if $p = (3, 4)$.

Additionally, let S denote the set of *complete states* of the OTEFSM, that is, both time-related and not time-related state information. Formally:

$$S \overset{def}{=} Q \times (T \to \mathbb{N}^2_\infty)$$

with elements $(q, \tau) \in S$, and $\mathbb{N}_\infty$ denoting the set of natural numbers plus "infinity". Practically, the two natural numbers associated with each timer label represent the start and end of the time interval within which the expiry of the timer may occur. The pair (∞, ∞) marks a disabled timer.

For two timer states $\tau, \tau' : T \to \mathbb{N}^2_\infty$ a partial order $\leq$ shall be defined expressing interval inclusion, formally $\tau \leq \tau'$ iff $\forall \psi \in T \ ([\tau(\psi)_1, \tau(\psi)_2] \subseteq [\tau'(\psi)_1, \tau'(\psi)_2])$.

Furthermore, let $\perp$ be a special value $\perp \notin T \cup \Sigma$ to indicate nonexistence. The abbreviations T_0 and Σ_0 will be used for T or Σ, respectively, plus $\perp$.

Each transition $\delta \in \Delta$ is a quintuple $\delta = (\psi, e, \alpha, \theta, \pi)$:

- $\psi \in T_0$ determines the timer that has to expire to enable the firing of δ. In case $\psi = \perp$, transition δ is not time-dependent.
- $e \in \mathbb{P}(Q \times \Sigma \times \Sigma_0) \cup \mathbb{P}(Q \times \{\perp\} \times \Sigma_0)$ is the enabling predicate of transition δ. e defines both the conditions under which δ may or must fire and its input/output behavior. The first powerset above denotes transitions triggered by inputs, while inputless transitions are covered by the second. Only inputless transitions are allowed to be time-dependent.
- $\alpha : e \to Q$ is the state transformation function describing the simple state change effected by the firing of δ.
- $\theta : T \to \mathbb{N}^2 \cup \{(\infty, \infty)\}$ is a partial function that determines the effect of δ on the timer state. $\theta(\psi) = (a, b)$ means that timer ψ is started to expire after at least a and at most b time units, whereas $a = b = \infty$ is used to indicate the cancelling of a timer.
- $\pi \in \mathbb{N}$ is the *priority* of δ. Greater π means higher priority.

3.2 Observational Semantics

To define the observational semantics of an OTEFSM $M = (Q, T, \Sigma, \Delta, d)$, it is useful to start with a function $\Psi : \Delta \times \mathbb{N} \times S \times \Sigma_0 \to \mathbb{N}^2$ expressing an *activation priority*. $\Psi(\delta, t, s, i)$ denotes the activation priority of transition δ at time t given the complete OTEFSM state s and an input i to be consumed by δ (otherwise $i = \perp$). This priority is represented by a pair of numbers because it depends both on the transition type and on π_δ, with the total order given by $(a, b) < (c, d)$ iff $a < c \vee (a = c \wedge b < d)$. The definition of Ψ distinguishes the following transition types:

spontaneous $\Psi(\delta, t, s, i) = (1, 0)$ if $\psi_\delta = \perp \wedge \pi_\delta = 0 \wedge \exists o \in \Sigma_0((s, \perp, o) \in e_\delta)$
immediate $\Psi(\delta, t, s, i) = (4, \pi_\delta)$ if $\psi_\delta = \perp \wedge \pi_\delta > 0 \wedge \exists o \in \Sigma_0((s, \perp, o) \in e_\delta)$
input $\Psi(\delta, t, s, i) = (2, \pi_\delta)$ if $\psi_\delta = \perp \wedge \exists o \in \Sigma_0((s, i, o) \in e_\delta)$
timed $\Psi(\delta, t, s, i) = (1, \pi_\delta)$ if $\psi_\delta \neq \perp \wedge \tau_s(\psi_\delta)_1 \leq t < \tau_s(\psi_\delta)_2 \wedge \exists o \in \Sigma_0((s, \perp, o) \in e_\delta)$
timeout $\Psi(\delta, t, s, i) = (3, \pi_\delta)$ if $\psi_\delta \neq \perp \wedge t \geq \tau_s(\psi_\delta)_2 \wedge \exists o \in \Sigma_0((s, \perp, o) \in e_\delta)$
not activated $\Psi(\delta, t, s, i) = (0, 0)$ otherwise.

Next, define the local state transition relation $\lambda \subseteq S \times \Delta \times \mathbb{N}^2 \times \Sigma_0^2 \times S$, which specifies the effect of firing a transition on the complete OTEFSM state. It expresses that state (q, τ) is transformed into state (q', τ') by the firing of δ at a time between t_1 and t_2, consuming symbol i as an input and producing symbol o as an output (unless i respectively o are $\perp$). Introducing an arrow notation, the definition of λ is:

$$(q, \tau) \xrightarrow[i,o]{\delta, t_1, t_2} (q', \tau') \text{ iff}$$

$$\forall t \in [t_1, t_2](\Psi(\delta, t, s, i) > (0, 0)) \wedge q' = \alpha_\delta(s, i, o)$$

$$\wedge \quad \forall x \in T : \tau'(x) = \begin{cases} (t_1 + \theta_\delta(x)_1, t_2 + \theta_\delta(x)_2) & \text{if } x \in \mathrm{dom}(\theta_\delta) \\ (\infty, \infty) & \text{if } x = \psi_\delta \wedge x \notin \mathrm{dom}(\theta_\delta) \\ \tau(x) & \text{otherwise} \end{cases}$$

The second case of the timer state change expresses that a timer is automatically disabled after it has triggered a transition.

Now the global state transition relation λ^* can be formulated, which extends λ to obey the prioritization imposed by Ψ. Its functionality is $\lambda^* \subseteq S \times \mathbb{N}^2 \times \Delta \times (\Sigma_0 \times \mathbb{N})^2 \times S \times \mathbb{N}^2$. It means that from state s entered at any time between t_1 and t_2 the next occurrence of a transition may be the firing of δ at any time between t_1' and t_2' resulting in the subsequent state s', consuming symbol i with its time stamp t_i and producing symbol o with its time stamp t_o. Again, $i = \perp$ or $o = \perp$ means no observable input respectively output, in which case the respective time stamps are irrelevant. With the abbreviation $P = \Psi(\delta, t_2', s, i)$, the relation λ^* (in arrow notation) is defined by:

$$\begin{aligned}
& s, t_1, t_2 \xrightarrow[(i,t_i),(o,t_o)]{\delta} s', t_1', t_2' \text{ iff } s \xrightarrow[i,o]{\delta, t_1', t_2'} s' \\
\wedge \quad & t_1 \leq t_2 \wedge t_1 \leq t_1' \leq t_2' \wedge t_2' \leq t_o \leq t_1' + d \wedge i \neq \perp \Rightarrow t_1' = t_2' = t_i \\
\wedge \quad & \forall \delta^* \in \Delta(\Psi(\delta^*, t_2', s, i) \leq P \wedge \forall t^* \in [t_1, t_2'[\\
& (\Psi(\delta^*, t^*, s, \perp) < (2,0) \wedge \psi_{\delta^*} = \psi_\delta \Rightarrow \Psi(\delta^*, t^*, s, \perp) < P))
\end{aligned}$$

From the viewpoint of executing an OTEFSM specification, this means that transitions may be triggered either solely by their state-dependent enabling condition (*spontaneous* and *immediate* transitions) or by the arrival of an input PDU (*input* transitions) or by the expiry of a timer (*timed* and *timeout* transitions). *Spontaneous* transitions have "may fire" semantics, i.e., their enabling does not enforce their firing, whereas *immediate* transitions must fire as soon as they become enabled, suppressing all other transition types. Triggered *input* transitions have "must fire" semantics as well, but may be overruled by *immediate* transitions. Time-dependent transitions have to be regarded as "must fire", too, with the actual timer expiry time drawn randomly from the expiry interval $[t_1, t_2]$ of the respective timer. If the enabling predicate of such a transitions becomes true *after* timer expiry, the transition is referred to as of the *timeout* type, overruling *input*, but not *immediate* transitions. From the viewpoint of trace analysis, this behavior of time dependent transitions has to be equivalently modeled as a "may fire" condition within the timer expiry interval and a "must fire" rule after its upper bound t_2, because the actual expiry time remains unknown.

Among triggered transitions of equal type, prioritization occurs according to π_δ. If this leaves more than one transition enabled, one is selected nondeterministically. While the firing of *input* transitions is always tied to the time stamp of the respective input PDU, output observations may be delayed by at most d time units after their generation.

An observation is a permissible protocol trace iff there exist an initial state $s^0 \in S$ and a sequence of globally permissible transitions of the form

$$s^0, 0, 0 \xrightarrow[(i^1,t_i^1),(o^1,t_o^1)]{\delta^1} s^1, t_1^1, t_2^1 \xrightarrow[(i^2,t_i^2),(o^2,t_o^2)]{\delta^2} \ldots \xrightarrow[(i^l,t_i^l),(o^l,t_o^l)]{\delta^l} s^l, t_1^l, t_2^l$$

that consumes and produces the observed symbols with their proper time stamps (again, $\bot$'s ignored for inputless respectively outputless transitions). This concludes the definition of the OTEFSM model.

3.3 Computability

The OTEFSM definition contains a number of subtle features that are both necessary and sufficient to facilitate the efficient computability of trace analysis. The following lemma describes the most important of them.

Lemma 1 *Assume there are a state transition*

$$(q, \tau), t_1, t_2 \xrightarrow[(i,t_i),(o,t_o)]{\delta} (q', \tau'), t'_1, t'_2$$

permitted by λ^* and a complete state $(q, \hat{\tau}), \hat{t}_1, \hat{t}_2$ such that $\hat{\tau} \geq \tau \wedge [\hat{t}_1, \hat{t}_2] \supseteq [t_1, t_2]$. Then,

a) *there exists a firing interval $[\hat{t}'_1, \hat{t}'_2] \supseteq [t'_1, t'_2]$ such that*

$$(q, \hat{\tau}), \hat{t}_1, \hat{t}_2 \xrightarrow[(i,t_i),(o,t_o)]{\delta} (q', \hat{\tau}'), \hat{t}'_1, \hat{t}'_2$$

b) *for any λ^* transition satisfying **a**, $\tau' \leq \hat{\tau}'$.*

Outline of proof: By the definition of λ, the wider timer expiry intervals in $\hat{\tau}$ and the wider expansion start interval $[\hat{t}_1, \hat{t}_2]$ permit a superset of firing times compared to τ and $[t_1, t_2]$, because all transitions being λ-activated according to τ and $[t_1, t_2]$ are as well λ-activated according to $\hat{\tau}$ and $[\hat{t}_1, \hat{t}_2]$ and overruling by *timeout* transitions does not occur earlier (refer to the definition of Ψ). The firing time does not influence the simple state transformation $q \rightarrow q'$ at all. Therefore at least $[\hat{t}'_1, \hat{t}'_2] = [t'_1, t'_2]$ provides a valid transition, which proves **a**. **b** is a direct consequence of the way the resulting timer states τ' and $\hat{\tau}'$ are computed according to λ. $\square$

Theorem 1 *Trace analysis against an OTEFSM specification is computable, provided that the initial state s^0 is known and the length of possible transition sequences without input and output is bounded by the specification.*

Outline of proof: For a permissible observation (I, O) there exists an explaining λ^* action sequence A. By the definition of λ^*, the enabling times of any transition to be the next to fire in any intermediate step form a single coherent interval. Consequently there is a *largest* enabling time interval $[t^k_1, t^k_2]$ unambiguously defined for any action in A. By iteratively substituting these largest enabling intervals into A, a normalized action sequence A_n is unambiguously determined that is a valid λ^* expansion and also explains (I, O), as can be concluded by induction from lemma 1. Since the intermediate states s^k are deterministically determined[*] by the function α_δ, T and the number of observed symbols are finite and the number of transitions per symbol is bounded by assumption, the normalized action sequences that might explain (I, O) can be enumerated by exhaustive state space search. $\square$

[*]Note that the output message of δ need *not* be deterministically determined, as it is required in [EvB95], since e_δ is a relation.

4 UNCERTAIN TRACE ANALYSIS

In order to overcome the known-initial-state restriction, the *FollowSM* method makes the trace analysis capable of processing *uncertain state descriptions*. This means the state space search algorithm is generalized to operate on range subsets rather than on single values of state variables. This section explains the extension of basic trace analysis to cover uncertainty.

For an informal outline, consider a state space of $(a, b) \in \{0, 1, .., 255\}^2$. The initial state description is $s^0 \in (0..255, 0..255)$. Let δ be a candidate transition defined by an OTEFSM specification. Assume its enabling predicate e_δ defines an associated output message $\sigma(x)$ with $x \in \{0, .., 255\}$ and requires $a < x$, and the transformation function α_δ performs $a \leftarrow a + 10$; $b \leftarrow 17$. Assume the next observed output message is $\sigma(42)$. Trace analysis based on a straightforward reference implementation cannot make any use of this information. However, *FollowSM* will instantiate the enabling predicate to $a < 42$, restrict the given state description s^0 to the subset enabling δ, and apply δ speculatively:

$$s^0 \in (0..255, 0..255) \xrightarrow{a<42} s' \in (0..41, 0..255) \xrightarrow{a \leftarrow a+10;\, b \leftarrow 17} s^1 \in (10..51, 17)$$

s^1 is – after considering just a single transition – a much more certain description of a possible state than s^0 has been. Of course, all other transitions have to be considered for speculative execution as well, and quite a lot of them may be enabled by s^0. In fact, at least all transitions without associated interactions will be.

However, this kind of uncertain state expansion cannot be reasonably performed in a complete and correct way, i.e., yielding a “pass” if and only if there exists a permissible action sequence for an observation. This is due to space and time limitations, especially due to the fact that the satisfiability of general boolean expressions is not efficiently computable. Therefore, the problem is solved approximately, obeying these two requirements:

1. No permissible trace may be rejected as a “fail”. That is, the algorithm shall be *correct*, but not necessarily *complete*, with respect to trace rejection.
2. As soon as state space search has reached a “certain” state description, further expansion of this state has to be both correct and complete.

Formally, first a *representation space* R has to be chosen for a state space Q such that at least the one completely uncertain and all completely certain state descriptions are representable:

$$R \subseteq \mathbb{P}Q \setminus \emptyset \quad \wedge \quad Q \in R \wedge \forall q \in Q(\{q\} \in R)$$

Let further $U = R \times (T \to \mathbb{N}^2_\infty)$ denote the set of *uncertain complete state descriptions*. Let $\mathrm{st} : U \to \mathbb{P}S$ be a function to directly refer to those certain complete states covered by the argument, i.e., $\mathrm{st}(r, \tau) \overset{def}{=} \{(q, \tau) | q \in r\}$

Uncertain trace analysis can now be specified as an extension of λ^* to the relation μ^*, substituting U for S in its signature and u for s in its arguments, in this way: With

$$v' \overset{def}{=} \left\{ s' \in S \,\middle|\, \exists s \in \mathrm{st}(u) : s, t_1, t_2 \xrightarrow[(i,t_i),(o,t_o)]{\delta} s', t'_1, t'_2 \right\}$$

collecting those states actually reachable from states in u,

$$u, t_1, t_2 \xrightarrow[(i,t_i),(o,t_o)]{\delta} u', t'_1, t'_2 \text{ iff}$$
$$t_1 \le t_2 \;\wedge\; t_1 \le t'_1 \le t'_2 \;\wedge\; t'_2 \le t_o \le t'_1 + d \;\wedge\; i \ne \bot \Rightarrow t'_1 = t'_2 = t_i$$
$$\wedge \;\; \mathrm{st}(u') \supseteq v' \;\wedge\; (|\,\mathrm{st}(u)| = 1 \Rightarrow \mathrm{st}(u') = v')$$

In uncertain trace analysis, an observation is considered a permissible protocol trace iff there exist a sequence of transitions permitted by μ^* of the form

$$u^0, 0, 0 \xrightarrow[(i^1,t_i^1),(o^1,t_o^1)]{\delta^1} u^1, t_1^1, t_2^1 \xrightarrow[(i^2,t_i^2),(o^2,t_o^2)]{\delta^2} \ldots \xrightarrow[(i^l,t_i^l),(o^l,t_o^l)]{\delta^l} u^l, t_1^l, t_2^l$$

that consumes and produces exactly the observed symbols with their proper time stamps, with $u^0 = (Q, T \times \{(0,\infty)\})$ being the initial state description denoting complete uncertainty.

Above requirements 1 and 2 will now be investigated by theorems 2 and 3 respectively.

Theorem 2 *Every observation permissible according to basic trace analysis is permissible according to uncertain trace analysis.*

Outline of proof: It has to be shown that for any λ^* expansion path there exists a μ^* expansion path starting from u^0 yielding the same observations. As an inductive hypothesis consider some state $s = (q, \tau) \in S$ and some corresponding state description $u = (r, \tau_u) \in U$ that fulfill $q \in r \wedge \tau \le \tau_u$. Further assume some valid λ^* state transition

$$(q, \tau), t_1, t_2 \xrightarrow[(i,t_i),(o,t_o)]{\delta} (q', \tau'), t'_1, t'_2$$

It follows from lemma 1 that for some $\tau'_u \ge \tau'$

$$(q, \tau_u), t_1, t_2 \xrightarrow[(i,t_i),(o,t_o)]{\delta} (q', \tau'_u), t'_1, t'_2$$

is also valid. Consequently, by the definition of μ^* there exists a valid uncertain state expansion

$$(r, \tau_u), t_1, t_2 \xrightarrow[(i,t_i),(o,t_o)]{\delta} (r', \tau'_u), t'_1, t'_2$$

such that $q' \in r'$, which completes the inductive step. Since the fixed u^0 for uncertain and any s^0 for basic trace analysis fulfill the inductive hypothesis, the theorem follows by induction over the length of the expansion path. □

Theorem 3 *Basic and uncertain trace analysis are equivalent provided an initial state description $s^0 = (q^0, \tau^0)$ resp. $u^0 = (\{q^0\}, \tau^0)$ without uncertainty is given.*

Outline of proof: It suffices to show that singleton states are always expanded into singleton states again by μ^*, formally that $|v'| = 1$ whenever $|\,\mathrm{st}(u)| = 1$. Then, μ^* and λ^* become obviously equivalent by their definitions. $|v'| = 1$ is valid for singleton state descriptions because the resulting state s of an expansion step is deterministically defined by α_δ, ϕ_δ, and λ if the transition δ, the firing interval, and the observable interactions are given. □

Theorem 4 *Uncertain trace analysis against an OTEFSM specification is computable, provided that the length of possible transition sequences without input and output is bounded by the specification.*

Outline of proof: This proof works quite similar to that of theorem 1. Here, a normalized explaining action sequence A_n is derived by setting all intermediate simple states to $Q \in R$ and, again, iteratively selecting the maximum firing-time intervals as determined by the timer states τ^k, the current time intervals $[t_1^k, t_2^k]$, and the interaction time stamps t_i^k, t_o^k according to λ. By the definition of μ^*, such A_n is permissible. It is unambiguously determined because the resulting timer states are. It also explains the observation because of part b of lemma 1. Thus, computation is possible by exhaustive search for such a normalized action sequence explaining the observation. $\Box$

5 IMPLEMENTATION

This section deals with the actual implementation of OTEFSM-based uncertain trace analysis and describes some properties of the *FollowSM* prototype.

One additional issue omitted in the formalization is that the first observed output messages might have been produced before the start of the analysis, such that the analysis algorithm must independently consider to ignore any prefix of the output sequence up to the maximum output delay d. This can be done easily because the number of such alternatives is finite and mostly small.

The proof of theorem 4 shows that the specification of uncertain trace analysis is extremely weak: It only ensures the validation of the timing constraints, but not at all the checking of the intermediate simple state descriptions $r^i \in R$ (with $u^i = (r^i, \tau^i)$). This weak specification is necessary to allow an implementation to use the selected representation space R for any intermediate results as well, e.g. during the evaluation of complex enabling predicates. The convergence of the state refinement process depends on the selected representation space, implementation details and the given protocol. It cannot be deduced from the given specification.

In practice, the representation space R is constituted by the combination of representation spaces individually selected for each state variable. In the implementation of *FollowSM*, several data types exist for state variable representations, which differ in the respective tradeoff between exactness and computational cost.

5.1 Search Strategy and Complexity

The theoretical results imply that OTEFSM based trace analysis is of exponential worst-case time complexity because only the number of branches in each step is bounded. But this is not of any practical relevance provided that the search strategy for the state space search is suitably chosen, as will be explained now.

The *FollowSM* implementation uses breadth-first search, in which it is different from the approaches in [BvBDS91] and [EvB95] where depth-first search is preferred. The decisive advantage of breadth-first search in the context of *FollowSM* is that it allows the algorithm to detect and make use of the reunification of state expansion paths: Whenever a state description is derived that equals or includes or specializes a previously generated one, that expansion path with the less general state description can be abandoned without loss of correctness. Therefore, the search paths expanded in parallel are synchronized according to their current time components. Figure 3 depicts an example where half of the search tree (dotted) turns out to be redundant because of a single path unification at a and b. The practical benefit from this solution is that there is some effective bound on the number of simultaneously open nodes in the search tree. Consequently, the trace analysis can be computed in almost linear time, regardless whether the given trace is valid or not. Furthermore, the space needed to store the open search nodes does not depend on the length of the trace, as it is the case in depth-first search with backtracking. Finally, the algorithm processes the trace in a message-by-message manner without jumping back and forth in time.

In short, breadth-first search with path unification appears to be the method of choice for on-line and real time trace analysis.

5.2 Derivation of Diagnostics

The diagnostics *FollowSM* can provide for the user are based on the reconstruction of a single sequence of internal actions that explains the observed trace. This single action sequence is derived from the search graph, which is often complicated and highly branched. Whenever path unification according to section 5.1 has occurred, the selection of the explaining action sequence becomes ambiguous. If one of the unified state descriptions is more general, its path is chosen since only this path can explain *all* subsequent behaviors. If some unified state descriptions are equal, one path is arbitrarily chosen.

When *FollowSM* detects a protocol violation by failure to expand the latest search node generated so far, the error cause is automatically characterized according to the condition that made the state expansion algorithm fail. Seven types of failure causes are distinguished in this way, including *"Enabled immediate/timeout/input transition ... did not occur according to output observation"*, *"Enabled transition ... suppresses all others at expiry of timer ..."*, and *"No transition was enabled to produce/consume PDU ... at time ..."*.

A single implementation fault in the IUT can easily result in thousands of observed protocol violations that are not of any individual interest. Therefore, *FollowSM* can combine "similar" protocol violations into a single error class diagnosis and provide statistical information about the occurrence of protocol violations related to this error class. Ideally, each error class represents exactly one cause of errors, for instance a single fault in the IUT software. The decision which detected violations are similar is made depending on the failure cause as explained above and on the internal action sequence immediately preceding a protocol violation.

5.3 Prototype Architecture

The prototype has been implemented in C++. It shows the basic modular structure drawn in figure 4. The protocol specific parts, dashed in the figure, consist of the OTEFSM specification of the target protocol and the *decoder* module. All the logic

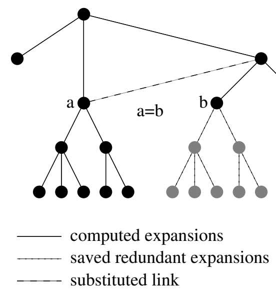

Figure 3 Reduction of complexity by path unification.

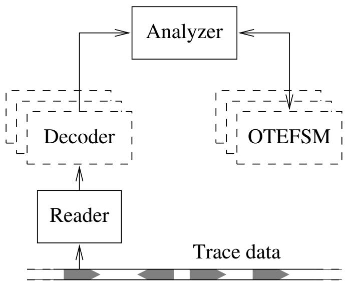

Figure 4 Basic architecture of the *FollowSM* prototype.

for the computation of the trace analysis and the handling of uncertainty is completely protocol-independent and implemented in the *analyzer* module. Only the very small *reader* module provides the interface to the trace data, so that it makes almost no difference whether the analysis is performed off-line, from a file, or on-line, connected to an actual communication channel.

Due to the breadth-first search strategy, *FollowSM* is able to run several independent state machines for different protocol layers and/or different virtual connections in parallel. A new instance of an OTEFSM is generated and initialized to an “unknown” state whenever a new connection identifier is found by the decoder.

6 EXPERIMENTAL RESULTS

The prototype implementation has been applied to two protocols of high practical relevance and considerable complexity: The protocol SSCOP according to the ITU-recommendation Q.2110 and the signaling protocol according to Q.2931, which constitute the layer 2 and 3 protocols respectively for the *Digital Subscriber Signalling System No. 2* (DSS2) on the *user-network interface* (UNI) in the broadband ISDN environment. The OTEFSM model of the SSCOP protocol consists of 138 transitions.

The most interesting experiment performed with the prototype was the on-line analysis of traffic between an active protocol tester running an executable SSCOP test suit (ETS) against a real SSCOP implementation. Overall, the results are very promising.

First, the convergence of the state descriptions to singleton states had to be confirmed by experiment (refer to section 5). It turned out that the trace analysis synchronizes with the communication after only a few messages – say between 3 and 10 depending on the currently executed protocol procedures. This does not hold for *all* state variables, because the SSCOP protocol includes state variables that are only involved in rare exceptional procedures. The values of such variables inevitably remain uncertain as long as they are not referred to. This does not prevent *FollowSM* from detecting those protocol violations that can be recognized from parts of the state description that do have converged. Figure 5 shows a small excerpt from the very beginning of an analysis report. There are three state variables included, namely the major state variable, a timer and a counter.

```
Time            Transition          PDU    PDU    State  Timer_CC         VT(CC)
                                    in     out
--------------------------------------------------------------------------------
0.0000                                            ?      ???              ???
0.0000          SendENDAK                  ENDAK  ?      ???              ???
0.0250          U3BGAK              BGAK          3..10  ???              ???
0.0250..0.0252  U4TimedOut                        1      off              4..-96
0.0250..0.0252  RequestConnection          BGN    2      0.9250..4.0252   1
0.0252          U2END               END           2      0.9250..4.0252   1
0.0747          U2BGREJ             BGREJ         1      off              1
0.0747..0.0749  RequestConnection          BGN    2      0.9747..4.0749   1
0.0749          ConnectionBegin     BGN           10     off              1
```

Figure 5 Example synchronization phase.

Second, during one experiment *FollowSM* detected an error in the ETS, which formulated an incorrect acceptance criterion for a message parameter. This resulted in a “fail” verdict from the ETS, while the *FollowSM* trace analysis correctly accepted the observation. On the other hand, *FollowSM* reported protocol violations for one of the two SSCOP implementations used in the experiments. These diagnostics could be validated manually and are consistent with the respective implementation’s being known to be faulty.

Third, the experiments delivered some information about the throughput of the on-line trace analysis with *FollowSM*. On a SUN Sparc5 workstation also running the *FollowSM* user interface and a standard UNIX operating system, SSCOP trace analysis was performed at a speed ranging between 200 and 400 transitions per second. This is, in fact, an order of magnitude that makes real time operation possible in many cases.

7 CONCLUSION

In this paper a new approach to protocol trace analysis has been presented. This approach meets all requirements of practical in-field use, including the checking of action timing and the ability to synchronize with the observed communication almost instantly and in any intermediate state.

The approach is based on the new concept of uncertain trace analysis and the new OTEFSM model for the specification of the target protocol. Both the OTEFSM model and uncertain trace analysis have been defined formally. Computability, correctness, and a restricted form of completeness have been shown for the presented approach.

Experiments with a prototype trace analyzer implementing uncertain trace analysis have proven that the method can be of high practical benefit. The trace analysis can even be performed fast enough such that real time on-line operation will be feasible in many cases.

Future work on the presented approach will focus on the automatic transformation of common protocol specifications, preferably in the language SDL, into OTEFSM models for uncertain trace analysis.

7.1 Acknowledgements

This work has been funded by Siemens AG, Germany. I would like to thank Dr. Wendisch and Dr. Xu of Siemens Communications Test Equipment for their help in making this publication possible. I am grateful to Ina Schieferdecker of German National Research Center for Information Technology, Research Institute for Open Communication Systems (GMD FOKUS) for the opportunity to perform the described experiments, and to my advisor, Günter Hommel.

REFERENCES

[A^{+}88] A. V. Aho et al. An optimization technique for protocol conformance test generation based on UIO sequences and rural chinese postman tours. In *Protocol Specification, Testing, and Verification, VIII* [IFI88], pages 75–86.

[Boc78] Gregor V. Bochmann. Finite state description of communication protocols. *Computer Networks*, 2:361–372, 1978.

[Bri88] Ed Brinksma. A theory for the derivation of tests. In *Protocol Specification, Testing, and Verification, VIII* [IFI88], pages 63–74.

[BvBDS91] O. B. Bellal, G. v. Bochmann, M. Dubuc, and F. Saba. Automatic test result analysis for high-level specifications. Technical report #800, University of Montreal, Department IRO, 1991.

[CCI87] CCITT. Recommendation Z.100: Specification and Description Language SDL. Contribution Com X-R15-E, CCITT, 1987.

[CL91] Samuel T. Chanson and Jeffrey K. H. Lo. Open systems interconnection passive monitor OSI-PM. In *Protocol Test Systems 3*, pages 423–442. University of British Columbia, 1991.

[EvB95] S. Alan Ezust and Gregor v. Bochmann. An automatic trace analysis tool generator for Estelle specifications. *Computer Communication Review*, 25(4):175–184, October 1995. Proceedings of the ACM SIGCOMM 95 Conference, Camebridge.

[FvB91] S. Fujiwara and G. v. Bochmann. Testing non-deterministic state-machines with fault coverage. In *Protocol Test Systems 4* [IFI91].

[IFI88] IFIP. *Proceedings of the IFIP WG 6.1 8th International Symposium on Protocol Specification, Testing, and Verification (1988)*, Amsterdam, 1988. North-Holland.

[IFI91] IFIP. *Protocol Test Systems 4, Proceedings of the 4th International Workshop on Protocol Test Systems*, Amsterdam, 1991. Elsevier Science Publishers, North Holland.

[ISO87a] ISO. ESTELLE: A formal description technique based on an extended state transition model. International Standard ISO/IS 9074, ISO, 1987.

[ISO87b] ISO. LOTOS: Language for the temporal ordering specification of observational behaviour. International Standard ISO/IS 8807, ISO, 1987.

[ISO91] ISO. OSI conformance testing methology and framework. International Standard ISO/IS-9646, ISO, 1991.

[IT94a] ITU-T. B-ISDN ATM Adaption Layer – Service Specific Connection Oriented Protocol (SSCOP). Draft new Recommendation Q.2110, ITU-T, 1994.

[IT94b] ITU-T. Digital Subscriber Signalling System No. 2 (DSS 2). User network interface (UNI) layer 3 specification for basic call/connection control. Draft new Recommendation Q.2931, ITU-T, 1994.

[JvB83] Claude Jard and Gregor v. Bochmann. An approach to testing specifications. *The Journal of Systems and Software*, 3:315–323, 1983.

[KCV92] M. C. Kim, Samuel T. Chanson, and Son T. Vuong. Protocol trace analysis based on formal specifications. In K. R. Parker, editor, *Formal Description Techniques (FORTE), IV*, pages 393–408. IFIP, North-Holland, 1992.

[LL91] D. Y. Lee and J. Y. Lee. A well-defined Estelle specification for the automatic test generation. *IEEE Transactions on Computers*, 40(4):526–542, April 1991.

[Sch94] Ina Kathrin Schieferdecker. *Performance-Oriented Specification of Communication Protocols and Verification of Deterministic Bounds of their Qos Characteristics*. PhD thesis, Technical University of Berlin, Department of Computer Science, November 1994.

[TKB91] J. Tretmans, P. Kars, and E. Brinksma. Protocol conformance testing: A formal perspective on ISO 9646. In *Protocol Test Systems 4* [IFI91].

[UP86] Hasan Ural and Robert L. Probert. Step-wise validation of communication protocols and services. *Computer Networks and ISDN Systems*, 11(3):183–202, March 1986.

[vBP94] Gregor v. Bochmann and Alexandre Petrenko. Protocol testing: Review of methods and relevance for software testing. In *Proceedings of the 1994 International Symposium on Software Testing and Analysis (ISSTA)*, ACM SIGSOFT Software Engineering Notes, Special issue, pages 109–124, August 1994.

AUTHOR'S BIOGRAPHY

Marek Musial received his diploma in computer science from the Technical University of Berlin in 1995. He did some work on genetic algorithms and participated in the development of an autonomous flying robot as the TUB entry to the 1995 International Aerial Robotics Competition in Atlanta, which took second place. Now he is working on his PhD thesis on ”intelligent” protocol monitoring and trace analysis in the real time systems group of Günter Hommel at the TUB Institute for Technical Computer Science.