On Fundamentals of the Trace Assertion Method

Michal Iglewski[a], Jan Madey[b], Krzysztof Stencel[b]

a. Département d'informatique, Université du Québec à Hull, Hull, Québec, Canada J8X 3X7
Email: iglewski@uqah.uquebec.ca Tel: (819) 773 1602 Fax: (819) 773 1638
b. Institute of Informatics, Warsaw University, Banacha 2, 02-097 Warsaw, Poland
Email: madey@mimuw.edu.pl, stencel@mimuw.edu.pl Tel: (48-2) 658 3165 Fax: (48-2) 658 3164

ABSTRACT

This paper is a step towards a formal description of the current version of the trace assertion method of module interface specification. We discuss some fundamental concepts and present a model for trace specifications together with an argument in favor of the expressiveness of the method. In particular, we explicitly distinguish two equivalence relations on the set of traces: one based on the observable features of an object, the other, defined by a trace specification of this object.

1 Introduction

Well designed software should be structured into *modules* according to the "information hiding" principle [11]. The *trace assertion method* (in short: TAM) [1, 15, 7] is a formal method for the abstract specification of interfaces of such modules. A module implements a number of homogenous, independent *objects*. A trace specification is a "black-box" specification, i.e., it describes only those features of an object that are *externally observable* and hides details of its internal structure. The method was first formulated in [1], and subsequently investigated by many researchers (cf. e.g. [9, 4]). A revised version of TAM was presented in [15]. Since then, the method has undergone many modifications [3, 7, 2, 16], yet there is no report that completely presents the syntax and semantics of the current version of TAM. The recent Ph.D. Thesis [16] is an important step in this direction.

The main goal of this paper is to discuss certain fundamentals of TAM and to propose a model for the method. In Section 2 we describe a restricted version of TAM. In particular, we distinguish two equivalence relations on the set of traces: one based on the observable features of an object, the other, defined by a given trace specification of this object. In Section 3 we define a Mealy machine which serves as a model for trace specifications. The rationale for some of our decisions is discussed in Section 4. Final conclusions and future plans are briefly presented in Section 5.

2 The trace assertion method

2.1 A characterization of an object

An entity is called an *object* iff it satisfies the following properties:

- it has *states*, produces *outputs,* can be affected by *events*,
- the sets of states, events, and outputs are finite,
- it is in one of its states at every instant of time,
- it may change its state only as the result of an event,
- outputs may be produced only in response to events,

- between an event and the output which is the response to this event, no other event may occur,
- both the next state and the produced output depend exclusively on:
  - the present state of this entity,
  - the event affecting this entity.

Additionally, we assume that at most one event may occur at any given instant of time.

2.2 Basic concepts

We are interested in describing an object from the point of view of an *external observer*, who can perceive only events affecting the object and the outputs produced in response to these events. In the remainder of Section 2 we assume that the object under observation is settled.

Let E be the set of events that can affect the object and R be the set of outputs that can be produced by the object. A *trace* is a finite sequence of pairs (E,O) where $E \in E$ and $O \in R$. We use the following notation for traces:

$$E_1.O_1.E_2.O_2. \ldots .E_n.O_n$$

Both subsequent pairs and components of each pair are separated by a dot, "**.**". The dot is also used as an operator defined on traces. If T_1 and T_2 are traces then $T_1.T_2$ is a trace obtained by concatenation of T_1 and T_2. The empty sequence is called the *empty trace* and denoted by "_" (the underscore). Notice that the empty trace is a neutral element of the dot operation.

Traces are intended to represent the externally visible behavior of the object. However, there exist traces that cannot be used for this purpose. In order to partition traces according to this criterion, we define the notion of feasibility. A trace $E_1.O_1.E_2.O_2. \ldots .E_n.O_n$ is *feasible* iff for each $i \in \{1,2,\ldots,n\}$ output O_i may be produced by the object in response to event E_i after sequence $E_1.O_1.E_2.O_2. \ldots .E_{i-1}.O_{i-1}$.

If T and $T.S$ are both feasible traces, then S is called a *feasible extension* of T. The set of all feasible extensions of T is called the object's *behavior after T*.

Two feasible traces are *observationally equivalent* iff the object's behaviors after these traces are the same. In other words, an equivalence relation (denoted by "$\overset{o}{\equiv}$") is defined on the set of all feasible traces, such that $T_1 \overset{o}{\equiv} T_2$ iff T_1 and T_2 have the same sets of feasible extensions. This relation is called the *observational equivalence relation*[1].

Note that if we know the relation "$\overset{o}{\equiv}$", then the description of the object's behavior after a feasible trace T also specifies the object's behavior after each trace from the equivalence class of T. This corollary is fundamental for TAM.

2.3 A trace specification of a module

As mentioned in Section 1, a module implements homogenous[2] objects and the state of each object in a module is independent of the states of other objects in this module. In the restricted version of TAM which we are dealing with, an event may alter only one object. Hence, we may assume without sacrificing generality, that there is only one, *generic*, object in a module.

Let us now describe how the terms introduced previously are used in applications of TAM. An object can communicate with the outside world by means of:

- a set of programs that can be used by objects from other modules to provide information to, and/or receive information from the object — they are called *access-programs*;
- a vector of external variables that the object observes — they are called *input variables*;
- a vector of variables whose values are computed by the object and can be observed externally — they are

1. The term *trace equivalence relation* is used in the previous descriptions of TAM. We will elaborate this issue in Section 2.4.
2. Homogenous objects have the same sets of feasible traces.

called *output variables.*

Thus, the following events can affect an object:

- access-program invocations,
- selected changes of values of the input variables, called *input variable events*[3];

and the following outputs can be produced by an object:

- values returned by access-program invocations,
- values of output variables[4].

In TAM we describe objects from an external observer's point of view. Therefore, we should specify which traces can be observed, i.e., which are feasible. Usually this is not an easy task — even in simple cases formulae characterizing such traces are complex. The authors of TAM have invented a simpler technique [4, 15] which is based on the corollary from Section 2.2. In this technique, however, it is not necessary to describe the observational equivalence completely. We have to present only a small subset of it. The following steps should be accomplished:

- We choose a subset of traces to be called *canonical traces.* Let us denote this set by C. We assume that the empty trace is a member of C. We describe the object's behavior only after traces from C.
- We define *the output relation, out*; it is a subset of the product $C \times E \times R$. This relation states which outputs can be produced in response to events after a canonical trace. $(T_C, E, O) \in out$ iff O can be produced in response to E after trace T_C. In other words, we define the set of feasible single-event extensions of canonical traces. The relation *out* has to satisfy the following property:

  $$\forall T \in C \ \forall E \in E \ \exists O \in R \ [(T, E, O) \in out].$$

  This means that for each canonical trace and each event at least one output should be specified.
- We define a subset of "$\stackrel{o}{\equiv}$" by means of the *extension function*, *ef*, which maps from *out* into the set of canonical traces. If $ef(T_C, E, O) = S_C$, then we know that $T_C.E.O$ is observationally equivalent to S_C and thus we may further assume that the object's trace is S_C.

The extension function allows us to reduce each feasible trace to an observationally equivalent canonical trace (this is discussed in the next section). Using the output relation we can predict which outputs the object will produce in response to events after a canonical trace. Therefore, the technique described above completely specifies the externally observable behavior of the object.

A *trace specification* consists of a description of the generic object from the module, and contains:

- a list of input and output variables,
- a definition of input variable events,
- a list of access-programs with a description of arguments and outputs produced by each access-program,
- a definition of the characteristic predicate of the set of canonical traces, *"canonical",*
- a definition of the extension function,
- a definition of the output relation.

2.4 The specification equivalence

The feasibility of traces may be deduced from the definition of the set of canonical traces, the extension function, *ef*, and the output relation, *out*, as follows.

Let us introduce a *reduction function, r,* that maps the set of feasible traces onto the set of canonical traces. We

3. It is reasonable to define events as only those changes of the input variables which influence the object's behavior and to ignore those which are irrelevant.
4. The characteristic of an object presented in Section 2.1 implies that the values of output variables may change only as a result of an event. Thus, it is sufficient to focus attention solely on the values of output variables just after events.

define it, together with the feasibility of traces (predicate "*feasible"*), by a parallel induction:

- *feasible*(_)
- $r(\_) = \_$
- $feasible(T.E.O) \Leftrightarrow feasible(T) \wedge (r(T),E,O) \in out$
- $r(T.E.O) = ef(r(T),E,O)$

Since a trace specification states which traces are feasible, it also determines the observational equivalence relation (cf. Section 2.2). The reduction function defines another equivalence relation on feasible traces, "$\overset{s}{\equiv}$". We call two feasible traces T_1 and T_2 *specification equivalent*, $T_1 \overset{s}{\equiv} T_2$, iff $r(T_1) = r(T_2)$. The predicate *canonical* should define unique representatives of equivalence classes of "$\overset{s}{\equiv}$". Otherwise, there exists a canonical trace U such that $r(U) \neq U$ and the specification is not correct.

For each pair of feasible traces, T_1 and T_2, it is true that $T_1 \overset{s}{\equiv} T_2 \Rightarrow T_1 \overset{o}{\equiv} T_2$. Usually this implication holds in both directions, i.e., the two relations on traces, "$\overset{s}{\equiv}$" and "$\overset{o}{\equiv}$", are identical; they are called the *trace equivalence relation*, and are denoted by "$\equiv$". Sometimes, however, "$\overset{s}{\equiv}$" may partition the set of feasible traces into smaller equivalence classes than "$\overset{o}{\equiv}$" does. Thus, there may exist different canonical traces that are observationally equivalent.

Let us illustrate the possible differences between the two relations using a simple, well-known example[5]. We have a limited capacity stack of non-negative integers with the following access-programs:

- PUSH(a) pushes a onto the top of the stack if the stack is not full, or changes nothing otherwise,
- POP removes the top element from the stack if the stack is not empty, or changes nothing otherwise,
- TOP returns the value of the top element of the stack if the stack is not empty, or –99 otherwise.

It seems natural to define canonical traces as all sequences of PUSH(a_i) of the length up to the capacity of the stack. Such a sequence represents the current content of the stack, the only relevant information. Both equivalence relations would be the same in this case.

Let us now slightly modify our example and require that the TOP access-program return the value taken modulo 256 (for non-empty stacks). The new specification may differ only in the description of the output relation where the value returned by TOP is defined. Although the two traces PUSH(5) and PUSH(261) are canonical, they are observationally equivalent — the only way to retrieve information from the stack is to call the program TOP which will return 5 in both cases. In other words, if we do not change the definition of the predicate *canonical* in the specification of our modified stack, then the two equivalence relations ("$\overset{s}{\equiv}$" and "$\overset{o}{\equiv}$") are not identical anymore.

In this particular case we could easily change the second specification by modifying the canonical traces to be sequences of PUSH(a_i mod 256), resulting in the two equivalence relations being the same. However, such a modification might be not so straightforward and/or reasonable in the case of more complex examples (cf. e.g. [8]).

3 Modeling a trace specification

3.1 Mealy machines

A definition of a Mealy machine, as introduced in [10], can be easily extended to cover a non-deterministic case. By a Generalized Mealy Machine we understand a tuple $M = (\Sigma, \Delta, Q, q_0, \delta, \lambda)$, where:

- Σ is a non-empty finite set, called the *input alphabet,*
- Δ is a non-empty finite set, called the *output alphabet,*
- Q is a non-empty finite set, called the *set of states,*

5. It would not be reasonable to use a complex example in this paper — it would require a detailed explanation of the specification syntax. One can find such a more realistic example in [8].

- $q_0 \in Q$ and is called the *initial state* of the machine,
- $\delta \subseteq Q \times \Sigma \times Q$ and is called the *transition relation,*
- $\lambda \subseteq Q \times \Sigma \times Q \times \Delta$ and is called the *output relation*.

In this paper we will use the shorter term "Mealy machine" to cover the general case.

When machine M is in state q and the input symbol is a, M moves to a state r allowed by δ, i.e., such that $(q,a,r) \in \delta$. During this state transition M produces a symbol o as its output, such that $(q,a,r,o) \in \lambda$. M starts in the state q_0. We assume that at least one transition is allowed for each state and each symbol from the input alphabet, and that for each transition there exists at least one symbol from the output alphabet that can be produced during this transition. More formally,

- $\forall q \in Q \ \forall a \in \Sigma \ \exists r \in Q \ [(q,a,r) \in \delta]$
- $\forall q \in Q \ \forall a \in \Sigma \ \forall r \in Q \ \exists o \in \Delta \ [(q,a,r) \in \delta \Rightarrow (q,a,r,o) \in \lambda]$.

For a machine M and for a sequence $E_1,E_2,\ldots,E_n$ of symbols from the input alphabet we can define a set of all sequences $O_1,O_2,\ldots,O_n$ that can be produced by M in response to $E_1,E_2,\ldots,E_n$. We denote this set by $M(E_1,E_2,\ldots,E_n)$. Two machines M_1 and M_2 are said to be *equivalent* iff they have the same input and output alphabets and for each sequence $E_1,E_2,\ldots,E_n$ of symbols from the input alphabet, $M_1(E_1,E_2,\ldots,E_n) = M_2(E_1,E_2,\ldots,E_n)$.

3.2 A trace specification as a Mealy machine

If the set of canonical traces of a certain specification is finite, this specification can be modeled by a Mealy machine as follows. Let *ef* be the extension function and *out* the output relation. Then:

- Σ is the set of all events that can affect the object;
- Δ is the product of two sets: the set of all possible outputs of access-programs and the set of all possible tuples of values of output variables; if there exists an event which produces no output, we add a special symbol **nil** to Δ; we assume that this symbol is produced in response to such an event;
- Q is the set of all canonical traces;
- q_0 is the empty trace;
- δ is the relation such that $(T,E,S) \in \delta$ iff there exists $O \in \Delta$ such that $ef(T,E,O) = S$ and $(T,E,O) \in out$;
- λ is the relation such that $(T,E,S,O) \in \lambda$ iff $ef(T,E,O) = S$ and $(T,E,O) \in out$.

The class of all Mealy machines that are models for trace specifications does not contain all Mealy machines. Each machine which is the model of a specification satisfies the following condition:

$$(T,E,S) \in \delta \wedge (T,E,R) \in \delta \Rightarrow S = R \vee \neg \exists O \in \Delta \ [(T,E,S,O) \in \lambda \wedge (T,E,R,O) \in \lambda]$$

i.e., the input and the output of the machine fully determine the state transition. This property of Mealy machines is called *output driven determinism*.

The choice of canonical traces is almost arbitrary (cf. Section 2.3). Thus if we select an infinite set we cannot use Mealy machine as a model. Below we show that it is always possible to choose a finite set of canonical traces; the only task left to a specifier is then to find the appropriate set.

Claim:

For each object the set of equivalence classes of the observational equivalence relation "$\underline{\underline{o}}$" is finite.

Proof:

Let us assume that we want to specify an object Θ. According to the assumption made in Section 2.1, Θ can only be in a finite number of states; let us denote this set of states by Ω. Given a feasible trace T of Θ we can indicate the set of states in which Θ can be after T. We denote this subset of Ω by $States(T)$. The set $States(T)$ need not be a singleton because Θ can be non-deterministic. We define a new relation on traces of Θ:

$$T_1 \approx T_2 \Leftrightarrow States(T_1) = States(T_2).$$

Since the equality of subsets of Ω is an equivalence relation, "≈" is an equivalence relation too.

Note that if $T_1 \approx T_2$ then $T_1 \overset{o}{\equiv} T_2$. This is because the future behavior of Θ depends only on its current state: if, after T_1, object Θ can be in state q and can produce $O_1,O_2,\ldots,O_k$ in response to the sequence of events $E_1,E_2,\ldots,E_k$, then, after T_2, Θ can also be in state q and can produce $O_1,O_2,\ldots,O_k$ in response to $E_1,E_2,\ldots,E_k$ as well.

In the above paragraph we have shown that $T_1 \approx T_2 \Rightarrow T_1 \overset{o}{\equiv} T_2$. Since Ω has only a finite number of subsets, the set of equivalence classes of "≈" is finite. This implies that the number of equivalence classes of "$\overset{o}{\equiv}$" is also finite and ends the proof.

One should be warned, however, that an object with n states may have (2^n-1) equivalence classes of "$\overset{o}{\equiv}$"; therefore, the number of states of the modeling machine could be also very large. This situation may occur when the specified object is non-deterministic.

Let us also note that for every specification, S, the number of equivalence classes of "$\overset{s}{\equiv}$" is equal to the number of states in the Mealy machine, M, modeling S. If the two relations "$\overset{o}{\equiv}$" and "$\overset{s}{\equiv}$" are the same for S, then M is the minimal machine among all output driven deterministic machines equivalent to M.

4 A rationale for certain decisions

4.1 The form of a trace

Together with the method, the form of a trace has also undergone change. In the report on TAM [15], a trace starts with the initial values of the output variables and ends with the last event. Subsequently, however, it has become clear that such a solution has some drawbacks. In recent works on TAM (e.g. in [7, 16]) a trace is already defined in the same way as in this paper.

Let us compare the two forms and justify our choice. In the approach described in [15], the values of the output variables are computed after the last event and are used to extend the trace when a new event arrives. Thus, the current values of the output variables form a part of the state of an object. We chose to add these values at the end of the trace in order to describe the record of the observation of an object by the trace alone. This change made it possible, in particular, to give a precise semantics of output variables in the case of non-deterministic modules (cf. [8]). We also excluded the initial values of the output variables from the trace, because these values are not computed in response to any event (cf. Section 2.2). We assume that the values of the output variables are undefined when we begin the observation of the object. Note also that if we had not excluded the initial values of output variables from a trace, the concatenation of two traces could not have been defined naturally.

In the approach described in this paper, to initialize the output variables one should create an access-program which must then be invoked to establish their initial values.

4.2 Extension functions vs. extension relations

As was pointed out in Section 3.2, Mealy machines serve as a good model for trace specifications. However, transitions in Mealy machines are described using relations, while in TAM we use functions to define changes of the canonical trace of an object. In the case of non-deterministic modules, it might seem natural to use *extension relations* instead of extension functions. We have investigated the need to introduce such a change in the method and decided not to, for the following reasons.

First of all, a trace expresses what an external observer knows of the behavior of the object. This knowledge does not change non-deterministically; one knows what has just been observed (we assume that one is able to notice every

aspect of the externally visible behavior of the object). Hence, the use of a function to describe extensions seems to be the proper choice.

Secondly, the extension function is a sufficient tool for describing every externally observable aspect of an object's behavior, also non-deterministic; there is no need then to introduce a relation instead (we have shown in Section 2.2 how to derive a trace specification of an object from its observable features).

Thirdly, extension functions facilitate automatic verification of implementations in a simpler way than extension relations would do. It is very important in various software tools supporting TAM that are under development (cf. e.g. [5, 16]).

5 Conclusions

Trace specifications of module interfaces have a relatively long history but in the last few years we categorize them within the so called "functional approach" to computer system documentation[6] [13, 14]. This approach is based on the premise that professional documentation of complex computer systems can be expressed in mathematical terms (functions, relations, sets, etc.). Such documentation methods have already been used in a number of applications (e.g. [12]). Lessons learned are that the approach needs to be further elaborated and that supporting software tools are indispensable. In particular, we need to improve the mathematical soundness of the methods. We hope that the present paper is a step in this direction.

We have tried to give a precise and clear characterization of the basic concepts in TAM and to provide an appropriate mathematical model for trace specifications. In particular, the distinction between the two equivalence relations in TAM ("$\overset{o}{\equiv}$" versus "$\overset{s}{\equiv}$") has never been made explicit in the publications on the method known to us. The usage of the single notation ("$\equiv$") was a source of misunderstanding. We hope that we have at last sufficiently clarified these issues.

In this paper we have discussed a restricted version of TAM. Reference [6] is a continuation of the present work and fundamentals of TAM for multi-object modules are discussed there. In both papers, however, only software modules are dealt with (since this is the most common and natural case). Yet TAM can also be used for hardware modules, as investigated for example in [3] — we can easily extend our treatment to cover both types of modules. There are also other aspects of the method which require further research, like the role of input/output variables. Some initial proposals of more precise definitions of these notions, as we understand them, can be found in [2, 8]. We hope that we are approaching the final phase of a complete description of the syntax and semantics of TAM together with useful tools supporting its practical applications.

Acknowledgements

This paper emerged from the work on [8] and Janina Mincer-Daszkiewicz made substantial contributions for which we are grateful. Marcin Kubica offered us many helpful comments on earlier versions of this paper. Our discussions with Yabo Wang on his Ph.D. Thesis [16] also inspired our research.

This work was partly supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), by the State Committee for Scientific Research in Poland (KBN, grant 8 S503 040 04), and by Digital Equipment's European External Research Programme (EERP PL-002).

References

1. Bartussek, W., Parnas, D.L., "Using Traces to Write Abstract Specifications for Software Modules", in: *Proceedings of 2nd Conference of European Cooperation in Informatics*, Lecture Notes in Computer Science, 65. Springer-Verlag, Venice, 1978.

6. The term "documentation" is used as a more general one than "specification".

2. Bojanowski, J., Iglewski, M., Madey, J., Obaid, A., "Functional Approach to Protocol Specification", in: *Proceedings of the 14th International IFIP Symposium on Protocol Specification, Testing and Verification, PSTV'94,* Vancouver, B.C., pp. 371-378.
3. Erskine, N., "The usefulness of the trace assertion method for specifying device module interfaces", *CRL Report* No. 258, McMaster University, CRL, Telecommunication Research Institute of Ontario (TRIO), Hamilton, Ontario, Canada, 1992.
4. Hoffman, D.M., "The Trace Specification of Communications Protocols", *IEEE Transactions on Computers*, Vol. C-34, No. 12, December 1985, pp. 1102-1113.
5. Iglewski, M., Kubica, M., Madey, J., "Editor for the Trace Assertion Method", in: *Proceedings of the 10th International Conference of CAD/CAM, Robotics and Factories of the Future: CARs & FOF'94*, Zaremba, M. (editor), OCRI, Ottawa, Ontario, Canada, 1994, pp. 876-881.
6. Iglewski, M., Kubica, M., Madey, J., "Trace Specifications of Non-deterministic Multi-object Modules", *Technical Report* TR 95-05(205), Warsaw University, Institute of Informatics, Warsaw, Poland,1995.
7. Iglewski, M., Madey, J., Parnas, D.L., Kelly, P.C., "Documentation Paradigms", *CRL Report* No. 270, McMaster University, CRL, Telecommunication Research Institute of Ontario (TRIO), Hamilton, Ontario, Canada, 1993.
8. Iglewski, M., Mincer-Daszkiewicz, J., Stencel, K., "Some Experiences with Specification of Non-deterministic Modules", *Technical Report* RR 94/09-7, Université du Québec à Hull, Hull, Québec, Canada, 1994.
9. McLean, J.D., "A Formal Foundation for the Abstract Specification of Software", *Journal of the ACM*, Vol. 31, No. 3, July 1984, pp. 600-627.
10. Mealy, G.H., "A method for synthesizing sequential circuits", *Bell System Technical Journal*, Vol. 34, No. 5, 1955, pp. 1045-1079.
11. Parnas, D.L., "On the Criteria to be used in Decomposing Systems into Modules", *Communications of the ACM*, Vol. 15, No. 12, December 1972, pp. 1053-1058.
12. Parnas, D.L., Asmis, G.J.K., Madey, J., "Assessment of Safety-Critical Software in Nuclear Power Plants", *Nuclear Safety* 32, 2, April-June 1991, pp.189-198.
13. Parnas, D.L., Madey, J., "Functional Documentation for Computer Systems Engineering. (Version 2)", *CRL Report* No. 237, McMaster University, CRL, Telecommunication Research Institute of Ontario (TRIO), Hamilton, Ontario, Canada, 1991; to appear in: *Science of Computer Programming.*
14. Parnas, D.L., Madey, J., "Documentation of Real-Time Requirements", in: Kavi, K.M. (editor), *Real-Time Systems. Abstraction, Languages, and Design Methodologies*, IEEE Computer Society Press, 1992, pp. 48-56.
15. Parnas, D.L., Wang, Y., "The Trace Assertion Method of Module Interface Specification", *Technical Report* 89-261, Queen's University, C&IS, Telecommunication Research Institute of Ontario (TRIO), Kingston, Ontario, Canada, 1989.
16. Wang, Y., "Specifying and Simulating the Externally Observable Behavior of Modules", (Ph.D. Thesis)*, CRL Report* No. 292, McMaster University, CRL, Telecommunication Research Institute of Ontario (TRIO), Hamilton, Ontario, Canada, 1994.