Verification of Human Performance Models through Formal Methods

Lauro Enciso
Avelino J. Gonzalez
Electrical and Computer Engineering Department
University of Central Florida
Orlando, Fl 32816-2450
lenciso@pegasus.cc.ucf.edu
avelino@pegasus.cc.ucf.edu

Abstract
This paper discusses a formal approach to the verification of human performance models. It addresses the problem of detecting specification and performance errors in human performance models and focuses on verifying the correctness, consistency, and completeness of the interaction in automated verifications of complex human performance models. A systematic investigation is described for determining whether the formal specification provides the necessary information to enable the verification of a human performance model, so as to enable a Verification Support Environment (VSE) to ensure the performance of a specified task successfully. The point of the methodology is to verify specification of human performance models, to prove conformance with the specification by using formal methods based on hybrid automata, timed automata, temporal logic of actions, and to automate the proofs by using the VSE tool. In summary, the purpose is to verify human performance models using formal methods integrated in the VSE tool. This paper describes the issues involved in this work.

1. Introduction
As intelligent systems technology matures, constructing formal simulations of human performance models for a variety of situations becomes more feasible and practical. Researchers have built many simulations of human performance models. However, verification of such models has always been difficult. Thus, verification of human performance models, even for simple paths needs to be explored for any given application. The lack of well-established techniques and tools to support verification of human performance models by using hybrid automata complicates the problem.

Researchers in the last several years have emphasized the development of automated testing tools to support the verification and validation of distributed simulations. These tools can provide developers and users with a direct means of determining whether a system is complete and consistent, and whether it represents the real world situation. This methodology permits deeper understanding of data, results, and system dynamics [Clarke et al 2001].

On one hand, the theory of hybrid systems is broad and addresses the verifications of engineering applications. On the other hand, application of hybrid systems to human performance models is very small, yet it is increasingly important in order to develop hybrid automata theory to underlie human performance models. In contrast, a comprehensive and consistent means for verification of human performance models does not exist. Nevertheless, a wide variety of verification techniques and tools of standard software have been proposed and tested. All the verification techniques of human performance models have significant limitations.

There are differences between formal verification of human behavior models and conventional software systems. One is that formal verification of human performance models are designed to emulate the real world behavior while software system are designed to perform functions that are not necessarily an emulation of the real world. Another difference is that human performance models model simulations that operate on the expert’s pre-specified knowledge. This means that emerging behaviors are compared with the pre-specified knowledge of the expert.

The objective of verification and validation of an intelligent system is to guarantee the resulting system will provide an answer, solution or behavior equivalent to what an expert in the field would say if given the same inputs. Usually, computers can be used to simulate many physical systems. Most of these fall on a definition of human intelligence where, artificial intelligence is nothing more than simulating the works of the human brain in a computer.

Intelligent systems represent models of domains that are difficult or impossible to model without human intervention. More specifically, intelligent systems do not directly model the domain itself, but rather model the human’s interpretation of the domain. Therefore, it logically follows that it should be human performance that serves as the benchmark for determining an expert system’s correctness, rather than the domain itself [Knauf, 2000].

2. Verification, Validation and Certification
In any human performance model production, it is necessary to understand precisely not only the concepts, scopes and limitations of verification, validation, and
performance models are studied in [O'Keefe et al, 1987; Green & Keyes, 1987; Motoi et al, 1982; Zlatareva, 1992]. Gonzalez and Barr [2000] clarified the terms of verification and validation of intelligent systems by using the following considerations.

An analysis of meaning of verification and validation and how they are implemented for human performance models are studied in [O’Keefe et al, 1987; Green & Keyes, 1987; Motoi et al, 1982; Zlatareva, 1992]. Gonzalez and Barr [2000] clarified the terms of verification and validation of intelligent systems by using the following considerations.

- Verification deals with satisfying specifications.
- Verification involves internal consistency, completeness and correctness of knowledge base
- Validations involves testing
- Validation compares the system to the real world.

### 2.1. Verification

Starting from the definition of verification as [Gonzalez and Bar, 2000] “verification is the process of ensuring that the intelligent systems (1) conforms to specifications, and (2) that its knowledge is consistent and complete within itself”. This means that intelligent systems must be based on its specifications, and also free of internal errors based on its consistency (free of conflicts, redundancy, circularity, and subsumptions), and completeness (reachability).

There exists many previous works in the literature about verification of conventional software systems, engineering software systems, rule-based systems, knowledge-base systems, and intelligent systems. However, little, if any discussion exists about the verification of human performance models.

#### 2.1.1. Specification

Specifications are documents that explicitly describe the functionality of the product; that is, what the product is supposed to do. It should also list a set of constraints that the product must satisfy. In other words, how the system meets its specifications and how the system is to be built. The specification document will include not only the inputs to the product and the required outputs but also the stipulations that the product must be able to handle correctly a wide range of assumptions [Schach, 2002].

Every specification document includes constraints that the product has to satisfy. Other constraints might include portability, which means that the product can run on other hardware under the same operating system. Reliability is another constraint, which is defined as “the ability of a system or component to perform its required functions under stated conditions for a specified period of time” [IEEE, 1990].

Intelligent systems simulate human intelligence, this means that it is necessary to point to one task (or set of tasks) performed by human, and specify that it can be replicated in a computer. In general, specifications in intelligent systems play a secondary role in terms of importance [Gonzalez and Barr, 2000].

#### 2.1.2. Consistency and completeness

Consistency and completeness is a state in which the knowledge base is free of internal errors. Conversely, inconsistency can be described as the presence of errors such as conflicts, redundancies, circularity, and assumptions. Incompleteness can be embodied in unreachability, dead-end modules, unneeded elements, and missing links. These imply that there is something else missing that should be added.

### 2.2. Validation

According to [Gonzalez and Barr, 2000], “Validation is the process of ensuring that the output of the intelligent system is equivalent to those of human experts when given the same inputs”. This means that verified system is compared with the expert perception of the real world. Questions are then directed to the intelligent system for a response of some type of input, which means testing. The researchers consider three aspects of the validation problem 1) How to ask the intelligent system whether its range of operations is well covered; 2) How to tell whether the responses it provides are individually equivalent; and 3) How to make a validity statement for the entire system.

According to [Tsuruta et al, 2000] multi-step validation method reduce the load on busy experts to validate intelligent systems considering the following steps.

- Automatic validation by using computers that includes methods of solving, test cases, and test executions. These results are analyzed statistically to see whether they satisfy the estimation criteria and are presented to Knowledge Engineer.
- Validation by Knowledge Engineer, who examines not only errors found in automatic validation, but also measures the optimality and stability limits and attaches his knowledge to be used as validation information by knowledge engineer and experts.
- Validation by experts who judge only the test results presented by the knowledge engineer.
- Aggregation step, where all validation results are automatically aggregated and documented. If a problem is found, the test manager asks for explanation to the expert or the KE who made the decision and prepares a final report.

### 2.3. Certification

Certification is a written document to guarantee a system or component that satisfies its specified requirements and is acceptable for operational use [Koopman, 2001]. Certification can be applied either to organizations or
individuals, tools or methods, and systems or products. [Storey, 1996].

Verification and validation are components of the certification process for any embedded system. To answer why a product need certification there are three principal reasons 1) legal reasons, 2) commercial reasons, and 3) to show competence in specific areas. The regulatory agency not only approves the verification plan submitted by the developer, but also the future changes to the methods to insure that certification will not be affected. A certification must be established early on the process [Tran, 1999].

Different government organizations and private organizations are responsible for the certification of different products based on standards that provides information about software [RTCA, 1992]. The standard ISO 9000 certification allows to continually improves the product or service [ISO 1999].

### 3. Finite State Machines (FSM)

A finite state machine is defined as a finite set of states \( S \), a finite set of inputs \( I \), a transition function \( T \) that specifies the next state given the current state and the current input, an initial states \( S_0 \), and a set of final states \( F \) [Schach, 2002]. Formally, a finite state machine is defined as a 5-tuple \( (S, I, T, S_0, F) \)

Where:  
- \( S \) : A finite nonempty set of states  
- \( I \) : A finite nonempty set of inputs  
- \( T \) : A function from \( (S \sim F) \times I \) into \( S \) called the transition function.  
- \( S_0 \in S \) : The initial state.  
- \( F \) : The set of final states, \( F \subseteq S \)  

The use of FSM is extensive in computing applications. A practical example of FSM implementation is a menu-driven user interface [Schach, 2002]. The display of a menu is a state, and entering an input at the keyboard or selecting an icon with the mouse is an event that causes the product to go into some other states. The transition rules have the form

**current state** [menu] and **event** [option selection] \( \Rightarrow \) **next state**

To specify a product, a useful extension of FSM is to add one more predicates \( P \) to the preceding 5-tuple; where each predicate is a function of the global state \( Y \) of the product.

Formally, the transition function \( T \) is now a function from \( (S \sim F) \times I \times P \) into \( S \). Transition rules now have the form.

**current state** and **event** and **predicate** \( \Rightarrow \) **next state**

A FSM defines a set of states, which represent the different actions that make up the behavior of an entity. It also defines a set of inputs and outputs, and a set of transfer functions, which cause transitions to new states when inputs to the state machine satisfy those functions [Calder 1993].

### 4. Computer Generated Forces

To be familiar with vehicle simulations is important to be clear about the terminology and meaning of intelligent systems and computerized entities. The **Intelligent Systems** are simulations that contain entities acting intelligently as if they were humans’ controlled. In a military training context, simulated intelligent entities are often referred to as **Computer Generated Forces**. This term is used to describe simulated entities (vehicles, aircrafts, etc.) that take place in the training scenario. The entities, can be partially controlled by human interaction (Semi-Automated Forces, SAFOR), or completely automated (Autonomous Intelligent Platforms, AIP).

There exist some advantages of using both SAFOR and AIPs. By using SAFOR, the actions performed by the entities can be adjusted by a human, whose interaction handles the correct behavior of the control-logic. SAFOR requires less control-logic, which makes the system easier to develop and increases code reusability. One disadvantage is that learning proficiency is affected by its operators’ performance. An AIP, on the other hand, handles all circumstances that can arise. This increases the demand on the knowledge acquisition, the choice of the paradigm used, and the implementation. The cost of operating the system is less than SAFOR system, because of no human interaction is needed to control the simulation [Norlander, 1999].

One limitation of CGF systems is that they are highly complex and generally difficult to use and learn. However, many simulations today are trying to model human performance models, follow the way humans are “thinking” (e.g., neural nets, etc).

There exist a numbers of CGF systems based on different implementations as Rule-based systems (SOAR, PRS), as Finite State Machines (ModSAF, CCTT-SAF, IST-SAF) and integration of several common techniques (BASIC AGENT) [Tambe et al, 1993].

### 5. Context-based Reasoning

Context-based reasoning was introduced by Gonzalez & Ahlers [1998] to represent and reason about human behavior. CxBR has proven to be an effective and efficient methodology for representing tactical human behavior in simulated entities.

Context-Based Reasoning is based on the idea that (1) any recognized situation essentially defines a set of actions or procedures that correctly address the situation, and (2) identification of a future situation can be simplified if all things that are probable to happen while under the present situation are restricted by the present situation itself [Gonzalez, 1994].

CxBR is defined on basis of contexts, the most representative item. Contexts are defined as classes, which encapsulate functions to execute certain behaviors or actions typical of such contexts, and a definition of
what to expect when in that context. Tactical knowledge is complex and large, which makes it difficult to handle efficiently. It can be structured in several levels, according to its generality and its importance, but it typically divided in three kinds of hierarchical contexts: Mission context, Main context, and Sub context.

The Mission contexts are at the highest level. A mission defines the objective of the AIP. It is the general definition of the objectives of a scenario [Gonzalez, 1994]. The Mission context not only defines the things to avoid during the mission being undertaken, but also describes the political environment under which the mission is carried out. No more than one Mission context will be active at any one time.

The Main contexts represent tactical operation undertaken as part of the mission. The Main contexts not only represent the main element of the CxBR representational paradigm, but also contains all the necessary information to operate the AIP under major situations. As Mission defines a set of Main contexts, in which one of them is considered the default-context, an active context whose premises must always be satisfied. The default-context gives the AIP a fundamental behavior upon which it can always fall back. The AIP must always have an active context to drive the reasoning process [Norlander, 1999].

The Sub-context is a lower level tactical operation, typically carried out as part of a main context. Sub-Contexts are finite in duration and they are associated with one or more main contexts. A main context can be divided in a number of sub-contexts to reduce complexity without limiting the number of levels of contexts. Realistically speaking, only a limited number of behaviors can occur in any situation [Gonzalez and Ahlers, 1998].

6. Verification Support Environment (VSE)

VSE is an integrated system that follow a top-down design methodology based on structured, formal specifications and refinement used to prevent danger from faulty computer systems in safety-critical areas (traffic, control, and medicine). The higher levels of security require the use of formal methods. [Foster et al, 2000]. In VSE, functional, operational, specification, modeling, and deduction methods have been integrated within a system. The VSE system was developed for the consortium of German universities and industry supported by the German Information Security Agency (GISA) to support formal development process [Hutter et al, 1999].

The VSE system is a tool for formal development of software systems. It consists of: 1) a basic system for editing and type checking specifications and implementations written in the specification language VSE-SL, 2) a facility to display the development structure, 3) a theorem prover for treating the proof obligations arising from development steps, 4) a central database to store all aspects of the development, and 5) automatic management of dependencies between development steps [Hutter et al, 2000].

VSE uses the structure of given specification (parameterization, actualization, enrichment, or modularization) to distribute the deductive reasoning into local theories [Rock, 1999]. Each theory is an encapsulated unit, which consists of its local signature and axioms. Relations between different theories are represented by different links between theories. Each theory maintains its own set of consequences or lemmata obtained by using local axioms and other formulas included from linked theories [Hutter et al, 1999].

6.1. Formalisms in VSE

Formal development in VSE is based on two formalisms: abstract data types and temporal logic. The first one is used to specify data structures and functional computations, while a temporal logic is used to specify the dynamic behavior of systems with a persistent state. There is a fully developed methodology for abstract data types, used to provide values for state dependent variables in state based systems.

6.1.1. Theories

Theories in VSE represent classical mathematical reasoning [Stephan et al, 2000]. Formal verification implies the use of mathematical terms such as sets, functions, and relations, and a strictly mathematical reasoning (proofs). Mathematical preciseness increased safety from errors. A formal definition of syntax and semantics offers uniqueness on interpretation and objectivity. Reasoning model of formal calculi, guarantee correctness to a fixed set of rules of inference, which can be validated only once [Foster et al, 2000].

To specify the state transition systems, VSE uses a specification language close to Temporal Logic of Actions [Lamport, 1994]. In addition to the theory of compositional development [Abadi and Lamport, 1995], which covers the composition of systems using input and output variables; VSE supports shared variables by structuring operators.

6.1.2 Abstract Data Types

Abstract data types define a view of systems as collections of operations on certain sets of data. The syntax of a data type is given by a set of type of symbols. The signatures are interpreted by algebras. Specifications consist of a signature and a set of axioms, which describe the intended semantics. VSE supports recursive data structures like sets, lists, trees, queues, and arrays. In particular we consider freely generated structures where each element has a unique representation. VSE defines two ways of structuring data type specifications 1) importing several specifications enrichment and (disjoint) union to be modeled in VSE and 2) generic specifications providing an additional slot to describe the formal parameter including axioms [Hutter et al, 1999].

VSE implements a theory of data type refinement [Reif, 1992]. It is a translation between one level and the next. The refinement process is done by programs implementing the operations of the higher level (the export) in terms of the operations of the lower level
VSE supports the implementations by procedures of an imperative programming language [Foster et al, 2000].

The idea of modeling a system is based on state transition system, which is a collection of state transitions, called operations. State transitions system provide the abstract operation as an interface to their environment and hide their internal structure from direct access from outside. State objects not only can be considered as variables of an abstract imperative programming language, which uses abstract elementary operations, but also have to distinguished from logical variables, whose value does not depend on any state [Foster et al, 2000].

6.2. Deductive Support

VSE system offers deductive support for all concepts provided by the specification language (VSE-SL system). Verification plays an important part as a formal validation of abstract specifications. VSE verifies that the formal requirement specification satisfies a given security policy, or that an implementation is a refinement of the given specification. In real-time applications, many proof obligations can arise and each of these obligations must be verified in order to complete the formal development. However, the arising proofs can be too complex to be fully automatic. Thus there is a need for an integrated approach of interactive and automated theorem proving [Hutter et al, 1996]

The objective of deductive support is to minimize the time of verifying proofs. VSE provides interaction between human and machines in both directions. On the one hand the user acts as an “intelligent” heuristic to be used by the machine and on the other hand the machine provides a variety of high-level strategies to manipulate its behavior. To reduce complexity of specifications and arising proofs, VSE uses modularity. The deductive process is done within the development graph.

VSE is based on sequent calculi, which allows us to decompose complex proofs by application of calculus rules into smaller sub-problems on a structure preserving way, extending the partial proof tree. VSE incorporates various techniques of the sequent calculus to mimic such strategies.

The first source of inefficiency is the removal of existential quantifiers in Gentzen’s calculus. The next source of inefficiency is related to the problem of finding an appropriate sequence in which quantifiers of a sequence have to be eliminated [Hutter et al, 1996]. Since quantifier rules are usually not permutive, finding the correct order may be crucial for finding the overall proof. Taking ideas from the resolution calculus, VSE provides a calculus rule for a simultaneous elimination of several quantifiers.

6.3. Architecture

All information about the development process is maintained in a common project database, and the deduction systems and their databases are integrated into this administration system. Thus, the VSE system is split into two layers: The top layer has the administration system, and below are the layers of the deduction systems. The deduction system layer is itself divided into layers for the systems KIV [Heisel et al, 1989, 1990] and INKA (Biundo et al., 1986). All layers communicate with the user by a common interface. Information is automatically converted when switching between the layers [Hutter et al, 1996].

The VSE system consists of deduction components based on integrated versions of KIV (Karlsruhe Interactive Verifier) and INKA (Induction Prover Karlsruhe). The KIV system manipulates proofs involving programs, while INKA is used for first-order proofs, in particular for induction proofs in generated structures.

7. Temporal Logic of Actions (TLA)

There are a several number of formalisms for developing distributed, concurrent systems, which are based on model checkers and theorem provers. Many problems are associated with concerns of managing dependencies between specifications, refinements, and proofs for components of complex development. One of the major difficulties of distributed systems are mutual dependencies existing in such systems. The state-based system of VSE is based on a variant of TLA.

7.1. Stuttering Steps

State-based systems are used to model reactive and concurrent systems. Their semantics is given by behaviors, defined by an infinite sequences of states \( \overline{s} = s_0, s_1, \ldots, s_n \), where \( s \in S \) is a valuation of a data type. Therefore specifications of state-based systems import theories [Hutter et al, 1999]. The basic ideas of the VSE approach to state-based systems [Rock, 1999] are taken from TLA the Temporal Logic of Actions [Lamport 1994a]. System steps are specified by a collection of actions, which are first-order logic formulas that in addition to logical variables contain primed and unprimed flexible variables.

Temporal formulas \( \Phi \) describe sets of behaviors by using the temporal operators (always), and (often) [Manna & Pnueli, 1991]. The canonical form of specification of component [Stephan et al, 2000] is given by

\[
\exists x_1, \ldots, x_n (\phi_{init} \land [A_1 \lor \ldots \lor A_n]_\diamond \land \phi_{fair})
\]

The first order formula \( \phi_{init} \) defines the initial states while the actions \( A_i \lor \ldots \lor A_n \) describe the possible state transitions, and \( x_1, \ldots, x_n \): Internal variables

\( \phi_{fair} \): Stands for fairness requirements of the system
Stuttering index can be defined as \((A_1 \lor \ldots \lor A_n)_\pi\). This is an abbreviation of 
\[ (A_1 \lor \ldots \lor A_n \lor (x_1 = x_1' \land \ldots \land x_n = x_n')) \]
\(\pi\) consists of the output variables and local variables of a component (flexible variables). Stuttering means, that the corresponding behaviors are closed under the operation of changing sequences by adding or deleting finitely many steps where the states are not changed [Hutter et al, 2000].

Action has a particular structure to allow stuttering steps. \([A]_{\pi}\) is defined as \(A \lor (x_1 = x_1' \land \ldots \land x_n = x_n')\).

The stuttering index \(\pi\) mentions those variable guarantees not to change if a stuttering step occurs instead of A. \(\langle A \rangle_{\pi}\), stands for \(A \land (x_1 \neq x_1 \lor \ldots \lor x_n \neq x_n)\).

In general, a specification should be invariant under stuttering, meaning that adding or removing stuttering steps from a behavior does not affect whether the behavior satisfies \(\Phi\). It is easy to modify formula \(\Phi\), so it asserts to ensure that every step is either \(M\) step or a step that leaves \(x\) and \(y\) unchanged, the new definition is 
\[ \Phi = Init _{\Phi} \land (M \lor (x' = x) \land (y' = y)) \]

Two ordered pairs are equal if their components are equal, so conjunction \((x' = x) \land (y' = y)\) is equivalent to the single equality \(\langle x', y' \rangle = \langle x, y \rangle\) and also \(\langle x', y' \rangle\) as \(\langle x, y \rangle\). For any action \(A\), s.t \([A]_{\pi} \equiv A \lor (f^s = f)\)

### 7.2. Fairness

Fairness means that if a certain operation is possible, then the program must eventually execute it. The fairness requirements for a concurrent algorithm can be expressed in terms of WF (weak fairness) and SF (strong fairness) conditions by using enabled predicates. The weak fairness formula asserts that there are infinitely many \(\langle A \rangle_{\pi}\) steps in a behavior, or there are infinitely many states, in which \(\langle A \rangle_{\pi}\) is not enabled. The strong fairness formula \(SF_{\pi}(A)\) asserts that there are infinitely \(\langle A \rangle_{\pi}\) steps, or there are only finitely many states in which \(\langle A \rangle_{\pi}\) is enabled in a behavior. The Weak and Strong fairness conditions \(WF(A)\) and \(SF(A)\) for an action \(A\) are defined by [Hutter et al, 2000].

\[ WF_{\pi}(A) \equiv (\Diamond \langle A \rangle_{\pi}) \lor (\Diamond \neg Enabled \langle A \rangle_{\pi}) \]

\[ SF_{\pi}(A) \equiv (\Diamond \langle A \rangle_{\pi}) \lor (\Diamond \neg Enabled \langle A \rangle_{\pi}) \]

An action \(A\) is said to be enabled in a state \(s\) if there exists some state \(t\) such that the pair of states \(\langle s, t \rangle\) satisfies \(A\). The formula \(WF_{\pi}(A)\) asserts of a behavior that, if the action \(A \land (x_1 \neq x_1 \lor \ldots \lor x_n \neq x_n)\) ever becomes enabled and remains enabled forever, then infinitely many \(A \land (x_1 \neq x_1 \lor \ldots \lor x_n \neq x_n)\) steps occur. In other words, it is ever becomes possible and remains forever possible to execute \(A\) that changes \(x\) intuitively many such steps must occur.

The formula \(SF_{\pi}(A)\), where \(f\) is an action and \(A\) an action. This formula asserts that if \(A \land (x_1 \neq x_1 \lor \ldots \lor x_n \neq x_n)\) is enabled infinitely often, then infinitely many steps may occur. If an action is ever enabled forever, then it is enabled infinitely often.

### 8. Hybrid systems

Hybrid systems are real-time systems embedded in analog environments [Alur et al, 1997]. A common model for hybrid system is a hybrid automata defined as “finite graphs whose nodes correspond to global states” [Nannengart, 2000]. The discrete actions of intelligent systems are defined by a change of control locations. The continuous activities of the environment are defined by real-valued variables whose values change over time by differential equations.

To facilitate the verification process, we model the components of the human performance model hybrid system as hybrid automata. Hybrid systems are based on the transition systems of timed automata [Alur and Dill, 1994] with discrete and general continuous variables and the Integrator Computation Tree Logic.

The verification of human performance models through formal methods follows linear hybrid automata because the behavior of all variables in each location is governed by linear constraints of the instantaneous state changes.

Formal approaches automated by VSE will be used for the verification of human performance models.

### 8.1. Syntax

Formally, hybrid systems [Nannengart, 2000] can be defined as tuples of the form.

\[ H = (X, \ell, E, \text{dif}, \text{inv}, \text{guard}, \text{act}) \]

Where:

- \(X\) a finite set of data-value variables
- \(\ell\) a finite set of locations, defined as nodes of a graph
- \(E \subseteq \ell \times \ell\) is a finite set of transitions, edge of the graph with nodes from \(L\).
- \(\text{dif} : \ell \times X \mapsto CT\) is a mapping that associates each location and each data variable a CT over \(X\)
- \(\text{inv} : \ell \mapsto CF\) is a mapping that associates each constraint formula (CF).
- \(\text{guard} : E \mapsto CF\) is a mapping that associates each edges a CF.
- \(\text{act} : E \times X \mapsto CT\) is a mapping that associates each edge and each data variable a CT.
8.2. Semantics

A state of hybrid system \( (L, \phi) \) is a pair \((L, \phi)\) where \( L \subseteq \ell \) is a location and \( \phi : X \mapsto R \) is a valuation of the data variables. A state \((L, \phi)\) is admissible if \( \phi(\text{inv}(L)) \) hold [Nonnengart, 1999].

Given two admissible states \( \sigma = (L, \phi) \) and \( \sigma' = (L', \phi') \) we say that \( \sigma' \) is transition reachable from \( \sigma \) denoted by \( \sigma \mapsto \sigma' \) if there exists a transition \( T = (L, L') \in E \) with source \( L \) and target \( L' \) and both \( \phi(\text{guard}(T)) \) and \( \phi'(x) = \phi(\text{act}(T, x)) \) for each \( x \in X \).

We call \( \sigma' \), timely reachable from \( \sigma \) with delay \( \delta \) denoted by \( \sigma \mapsto^\delta \sigma' \), where \( \delta \) is a non-negative real number, if \( L = L' \) and for each \( x \in X \) there exists a differentiable function \( f_x : (0, \delta) \mapsto \mathbb{R} \), such that (1) \( f_x(0) = \phi(x) \) and \( f_x(\delta) = \phi'(x) \), and (2) for all \( \varepsilon \in \mathbb{R} \) with \( 0 < \varepsilon < \delta \):

Both \( \text{inv}(L)[x_1 / f_{x_1}(\varepsilon), \ldots, x_n / f_{x_n}(\varepsilon)] \) and \( f_x(\varepsilon) = \text{dif}(L, x)[x_1 / f_{x_1}(\varepsilon), \ldots, x_n / f_{x_n}(\varepsilon)] \) are true

\( \sigma' \) is timely reachable from \( \sigma \) denoted by \( \sigma \mapsto^\delta \sigma' \) if there exists a non-negative \( \delta \in \mathbb{R} \) such that \( \sigma \mapsto \sigma' \). \( \sigma' \) is said to be reachable from \( \sigma \) if \( (\sigma, \sigma') \in (\mapsto) \).

A run \( \rho \) of \( H \) with initial state \( \sigma_0 = (L_0, \phi_0) \) is a maximal sequence of states represented as \( \rho = \sigma_0 \mapsto_{f_0} \sigma_1 \mapsto_{f_1} \sigma_2 \mapsto_{f_2} \sigma_3 \mapsto_{f_3} \ldots \) where \( t_i \in \mathbb{R} \geq 0 \) and \( f_i = [0, t_i] \mapsto (X \mapsto \mathbb{R}) \), such that \( f_i(0) = \phi_i \), and moreover, \( \text{inv}(L_i[X / f_i(t)](X)) \) holds for all \( 0 \leq t \leq t_i \), \( (L_i(f_i(t_i)))) \mapsto \sigma_{i+1} \) and for all \( 0 \leq t' \leq t + \delta \leq t_i : (L_i, f_i(t')) \mapsto (L_i, f_i(t + \delta)) \).

The set of states contained in a run \( \rho \) is denoted as \( \text{States}(\rho) = \{ (L_i, f_i(i)) \mid i \in \mathbb{R}, 0 \leq t \leq t_i \} \).

The set of all runs of a hybrid system \( H \) with initial state \( \sigma \) is denoted by \( \text{runs}(H, \sigma) \).

A position \( \pi \) of a run \( \rho \) is a pair \( \rho = \sigma_0 \mapsto_{f_0} \sigma_1 \mapsto_{f_1} \sigma_2 \mapsto_{f_2} \sigma_3 \mapsto_{f_3} \ldots \) is a pair \( \pi = (i, r) \in \mathbb{N} \times \mathbb{R} \) such that \( 0 \leq r \leq t_i \). We denote the set of positions of a run \( \rho \) as \( \text{pos}(\rho) \).

Positions are ordered lexicographically, \( (i, r) \leq (j, s) \) if and only if \( i < j \) or \( (i = j) \) and \( r < s \). Also \( (i, r) \leq (j, s) \) if and only if \( (i, r) < (j, s) \) or \( ((i = j) \) and \( r = s) \) [Nonnengart, 1999].

By \( \rho(\pi) \) with \( \pi = (i, r) \) we denote the state \((L_i, f_i(r))\). Thus \( \text{States}(\rho) = \{ \rho(\pi) \mid \pi \in \text{pos}(\rho) \} \)

8.3 Example

A leaking gas burner example

The system has two states: in state \textbf{Leak}, the gas burner leaks; in state \textbf{Non-leak} is the non-leaking location. The variable \( z \) records the amount of time that the system has spent in the current state; it is used to specify properties (Leak and Non-leak). The clock \( x \) records the total elapsed time [Alur et al, 1995].

9. Conclusions

1. Verification by using formal methods appears to be an excellent paradigm for human performance models.
2. Verification of human performance models through formal methods is a process of applying requirements, specifications, proving, and automated verification of hybrid automata models for improving credibility.
3. Verification of human performance models through formal methods provides many benefits to artificial intelligence researchers because it emphasizes consistency, correctness, and completeness of the behavior.
4. Current V&V tools are difficult to use; it costs too much in time and budget, and they are hard to apply.
5. Automated verification of human performance models through formal methods can diminish some of the difficulties in applying current verification tools.

10. References
- [Lamport, L. 1994a], The temporal logic of actions, ACM Transactions on Programming Languages and Systems 16(3).