Conservative Approximations of Hybrid Systems*

Andrew K. Martin
Carl-Johan H. Seger
Integrated Systems Design Laboratory
University of British Columbia
Vancouver, B.C. V6T 1Z4 Canada

October 25, 1994

Abstract

Systems that are modeled using both continuous and discrete mathematics are commonly called hybrid systems. Although much work has been done to develop frameworks in which both types of systems can be modeled at the same time, this is often a very difficult task. Verifying that desired properties hold in such hybrid models is even more daunting. In this paper we attack the problem from a different direction. First we make a distinction between two models of the system. A detailed model is developed as accurately as possible. Ultimately, one must trust in its correctness. An abstract model, which is typically less detailed, is actually used to verify properties of the system. The detailed model is typically defined in terms of both continuous and discrete mathematics, whereas the abstract one is typically discrete. We formally define the concept of conservative approximation, a relationship between models, that holds with respect to a translation between specification languages. We then progress by developing a theory that allows us to build a complicated detailed model by combining simple primitives. Simultaneously, we build a conservative approximation by similarly combining pre-defined parameterized approximations of those primitives.

1 Introduction

In our research laboratory we have a computer controlled model train set consisting of about 35 feet of track, 13 computer controlled switches, three remotely controlled trains and approximately 60 position sensors. Although it was built to provide a test bed for designing mixed hardware/software systems, the train set has become a source of many challenging research questions related to hybrid systems. While some of the problems encountered are specific to our particular model train set, it is clear that others, such as old sensor data, noise, unreliable sensors and actuators are typical of other real systems as well.

One particularly challenging problem has been the development of a control system that guarantees freedom from collisions. Of course, there is a trivial solution to this task:

*This research was supported by a Killam pre-doctoral fellowship, by a postgraduate scholarship and by operating grant OGPO 109688 from the Natural Sciences and Engineering Research Council of Canada and by a fellowship from the B. C. Advanced Systems Institute.

never move any trains! The challenge, however, is to guarantee collision freedom while keeping the degree of utilization as high as possible. While working on this problem we encountered two difficulties. The first was formalizing the above problem statement. The second was verifying that a proposed solution was indeed correct. Partly because of our circuit verification background, and partly because of the nature of the specification, a model-checking verification approach seemed natural. However, such an approach would require a discrete state-machine model of the system.

Although, we could imagine various discrete state machine models of the train system, it was hard to be convinced that such models were accurate reflections of the physical reality. Moreover, we needed some way to rephrase the question "can two trains collide" into a verifiable property of such a state machine. Many aspects of the system, including what it means for two trains to collide are much easier to model using continuous mathematics. For example, the current position of an engine is, as a first approximation, a linear function of the speed of the train, its last known location, and the time that has elapsed since its location was known. On the other hand, any model of the system must also include the hard-wired digital control logic that is used to interface the computer with the train set. This control logic is modeled more naturally using discrete structures. While we needed to model the physical system using a combination of continuous and discrete mathematics, we had efficient verification procedures only for discrete state systems.

To capitalize on the strengths of both forms of representation, we have developed a theory of "conservative approximation." Loosely speaking, a discrete state model is a conservative approximation of a hybrid system if all verifiable safety properties of the discrete state model also hold in the hybrid system. One observation we made early in our effort to formalize this idea was that it is not enough merely to establish a correspondence between the two systems. Of equal importance is the establishment of a correspondence between the questions we can ask of one system and the questions we can ask of the other. Intuitively, we need some way of translating our "two trains do not occupy the same physical space" (which is a more precise statement of no-collision) question, to a question about the discrete state model of the system. Thus a conservative approximation must be defined in terms of a relationship between the questions, as well as a relationship between the models.

To prove that one model is a conservative approximation of another with respect to some question-translation, can be very difficult and time consuming. If we had to establish conservative approximation from the basic definitions every time we wanted to develop a model of a system, this approach would not be practical. To address this difficulty, we develop a calculus that allows us to build up conservative approximations of complex systems from many simpler approximations.

Section 3 develops a simple hybrid modeling framework. This framework is illustrated using a simplified version of our model train-set. The model is built by combining primitive components, such as integrators, multiplexors, and sampling devices. Section 4 gives a very general definition of conservative approximation. In Section 5, we develop a function that abstracts specifications. We then examine some of the primitives that were used in the train-set model, showing how to construct approximations of them that are conservative with respect to this function. Moreover, we show how to combine these primitive approximations to yield a conservative approximation of the entire train system. Finally, in Section 6, we show how to use the discrete conservative approximation and conventional model-checking to verify that trains do not collide. The paper introduces a moderate amount of notation to describe the trace-automata framework. This notation is summarized in Appendix A.

This paper should be seen as the first step towards building a practical system that would allow the user to construct natural and accurate models of physical plants by combining simple, well understood, components. At the same time as the model is constructed, the system would automatically construct a parameterized finite state conservative approximation. Desired properties of the physical system could then be verified against this approximation.

2 Related Work

The idea of conservative approximation has been presented in other contexts by several other researchers. Burch [4] constructs conservative approximations based on language homomorphisms. In his framework, a trace algebra is an algebra with composition and projection operators satisfying a small set of axioms. A trace is an element in the domain of such a trace algebra. Verification amounts to showing that the trace-set of an implementation is contained in the trace-set of a specification. Burch uses homomorphisms with respect to the operations in his algebras to construct mapping functions ψ_u and ψ_l from concrete domains to abstract ones. A verification problem is abstracted by mapping the specification with ψ_u and the implementation with ψ_l. The abstraction is conservative in that a successful concrete verification can be inferred from a successful abstract verification. Burch calls the pair (ψ_u, ψ_l) a conservative approximation.

The theory is applied to a variety of trace structures all of which associate varying amounts of timing information with discrete events. Burch shows how to construct conservative approximations from traces in which time is represented by real numbers, to traces in which time has discrete values. A second-method, based on power-set algebras over trace algebras, is used to construct conservative approximations from discrete time traces with explicit simultaneity, to traces with interleaving semantics. In both of these cases, the behaviour is represented as a sequence of discrete events, with real-valued time-stamps. Burch does not consider systems involving continuous traces of real time. Nor is the theory applied to hybrid models with multiple time scales.

Clarke *et al.* [5] describe an approach to abstraction in the context of finite-state transition systems. Programs and their abstractions are modeled as such systems. An abstraction is defined by a surjection from the concrete state-space to the abstract. Specifications are given in subsets of the temporal logic CTL. Atomic state formulae in the logic refer only to the abstract state. The abstraction surjection provides a natural interpretation of such formulae in the concrete domain. Thus the surjection provides both a translation for models, and a translation for specifications. The authors show that such abstractions are conservative when the specification language is limited to $\forall CTL^*$, a subset of CTL with only universal path quantification, and restricted temporal operators. A class of mappings are identified that define exact abstractions for CTL^*. A similar, but slightly more general approach to state abstraction is also described in [7].

Recently, there has been an increased interest in formalisms for describing the behaviour of hybrid systems. Raven *et al.* model hybrid systems using a real-time interval temporal logic [10]. Both specifications and their refinements are given as formulae in the logic. Verification amounts to showing that the refinement implies its specification. The logic is defined over interpretations in which states are viewed as functions over real-time. The technique is particularly aimed at expressing duration properties such as “Within any time period of length T, state S may occur at most c per cent of the time.” The logic is inherently undecidable, but some sound deduction rules are given

Alur *et al.* [1] use *hybrid automata* to model system behaviour. The state of a hybrid automaton consists of a location counter, drawn from a finite set of locations, and a set of real valued variables. The program counter defines the state of a finite state machine. Associated with each state is a set of differential equations, which govern the behaviour of the real-valued variables while the finite-state machine is in that state. Also associated with each state is a set of exceptions, predicates over the real-valued variables. Progress can be made by such an automaton in two ways. As time passes, the real-valued variables change their value according to the active set of differential equations. At any time, the entire state of the machine may change instantly according to a transition relation over both the location counter and the real-valued variables. To ensure progress, the automaton must make such an instantaneous transition before the elapse of sufficient time to satisfy one of the exception predicates.

The authors give a semi-decision procedure for proving that members of a restricted class of hybrid automata satisfy linear invariants over the real-valued variables. The procedures, based on computing and minimizing fixed points, are guaranteed to give correct results if they terminate. Several examples are given for which the procedures do indeed terminate.

A completely different approach to hybrid system specification [12] and verification [13] is given by Zhang and Mackworth. They use a formalism called constraint-nets to represent hybrid systems. Essentially, a constraint net represents the evolution of a system state as a set of mappings from algebraically defined time-structures to variable domains, which must have certain algebraic properties. These mappings represent the shared inputs and outputs of a set of transductions, and must be causally related with respect to the time structures. Semantically, a constraint net is denoted by the least fixed point of a set of equations. The existence of such a fixed-point is guaranteed by the algebraic properties of the time-structures and variable domains.

Kurshan and McMillan[6] pursue an approach that is similar to ours in the context of circuit analysis. They provide a well defined connection between analogue models of example circuits, and the languages accepted by discrete nondeterministic ω-automata. The connection is defined by a function called a *support map*, that maps from continuous functions of real time to ω-sequences of discrete symbols. Analogue properties of real circuits are modeled as systems of ordinary differential equations. A non deterministic ω-automaton is constructed so that the language that it accepts contains at least the image of every solution to these equations. Specifications are given as ω-automata in the discrete domain. A discrete model satisfies a specification if the language that it accepts is contained in that accepted by the task.

To a large extent the differences between the work of Kurshan and McMillan and this paper are matters of emphasis. For pragmatic reasons, circuit verification is generally performed using discrete models. However, the connection between these discrete models, and the analogue properties of the circuits is often unclear. Kurshan and McMillan's work address specifically this problem. They develop a particular method for modeling circuits as systems of differential equations, mapping solutions of such equations to discrete ω-sequences, and constructing ω-automata whose languages are (using our terminology) conservative approximations of the solutions of such equations with respect to this mapping. The authors suggest that larger circuits could be handled, by partitioning the system of differential equations, modeling the partitions, and composing the results, but this idea is not pursued in detail.

In contrast, we are interested in exploring a variety of ways in which continuous, discrete, and hybrid systems can be verified using conservative approximation. We have developed a very general framework in which such systems can be modeled. The framework supports decomposition in a very direct way, and much of this paper is devoted to a non-trivial example of this approach. The approach that we explore in this paper starts with a hybrid model of the system that is constructed hierarchically from simple components. Discrete conservative approximations of these components are then combined to form an approximation of the entire system, which can be verified using standard model checking techniques.

3 Trace-Automata

Our verification framework has two components: A set of models and a set of specifications. A model is used to represent the predicted behaviour of the system. A specification is used to represent the desired behaviour. The specifications are predicates over the set of models. Each model provides a context within which a truth value can be assigned to each specification. We begin by discussing the models, and postpone the discussion of specifications until the end of this section.

We present a set of models called *trace automata* in which the behaviour of hybrid systems can be described straightforwardly. The models are similar to the non-deterministic finite state automata that are traditional in computer science. *Trace-automata* are non-deterministic state-machines with multiple input tapes. While some of these input tapes contain the conventional finite-length strings, others contain $\mathcal{R}$*-traces* which are described below. The set of behaviours in which the system is capable of engaging is represented by the *language* accepted by such an automaton.

Let Σ be any, possibly infinite, set. Strings over Σ can be viewed as partial functions from the natural numbers to Σ. If $\Sigma = \{a, b, c, d\}$, for example, the string $\langle acbd \rangle$ maps the number 0 to a, 1 to c, 2 to b and 3 to d. To reinforce this view, we call such strings $\mathcal{N}$-traces. Formally, an $\mathcal{N}$-trace w of n symbols from the alphabet Σ is a function from the naturals less than n to Σ.

$$w : \mathcal{N}[0, n) \mapsto \Sigma$$

We denote the set of $\mathcal{N}$-traces of length n by Σ^n.

$$\Sigma^n \stackrel{\text{def}}{=} \{w : \mathcal{N}[0, n) \mapsto \Sigma\}$$

We denote the length, n, of an $\mathcal{N}$-trace, w, from Σ^n by $|w|$. The symbol ϵ denotes the single element of Σ^0. We use the notation $\Sigma^{\mathcal{N}}$ in place of the more traditional notation Σ^* to represent the set of all finite-length $\mathcal{N}$-traces over Σ.

$$\Sigma^{\mathcal{N}} \stackrel{\text{def}}{=} \bigcup_{n \in \mathcal{N}} \Sigma^n$$

Strings, or $\mathcal{N}$-traces as we call them from now on, are often used to represent the events that occur in a system over a period of time. Their use is particularly natural in frameworks for which time is discrete. Given that we are interested in developing a framework for describing continuous behaviours, it seems natural to extend the above definitions by replacing the naturals $\mathcal{N}$ with the non-negative reals $\mathcal{R}^+$. We call the resulting structures $\mathcal{R}$-traces. Just as an $\mathcal{N}$-trace maps from a left-open interval of the naturals to a set Σ, so an $\mathcal{R}$-trace maps from a left-open interval of the reals. The set of $\mathcal{R}$-traces of length r over Σ is defined for all positive reals r.

$$\Sigma^r \stackrel{\text{def}}{=} \{w : \mathcal{R}[0,r) \mapsto \Sigma\}$$

Σ^0 remains the singleton set $\{\epsilon\}$. Finally, we define $\Sigma^{\mathcal{R}}$, the set of all $\mathcal{R}$-traces over Σ.

$$\Sigma^{\mathcal{R}} \stackrel{\text{def}}{=} \bigcup_{r \in \mathcal{R}^+} \Sigma^r$$

Trace concatenation is denoted by juxtaposition. If $u \in \Sigma^a$, and $v \in \Sigma^b$ are Φ-traces — where Φ is either the naturals $\mathcal{N}$ or the reals $\mathcal{R}$ — then their concatenation, $uv \in \Sigma^{a+b}$, is the following partial function, illustrated here using lambda notation to bind the formal parameter.

$$uv \stackrel{\text{def}}{=} \lambda x \cdot \begin{cases} u(x) & \text{if } x \in \Phi[0,a) \\ v(x-a) & \text{if } x \in \Phi[a,a+b) \end{cases}$$

Observe that the concatenation operation is closed, associative, and has identity ϵ.

A trace-automata *domain* D is a triple $D = (\Pi, \Sigma, \Phi)$ in which Π is a set of distinct variables. Think of the elements of Π as the input tapes. With each variable $a \in \Pi$, Σ associates a domain of values Σ_a, while Φ associates a time-domain Φ_a, that must be either the reals $\mathcal{R}$ or the naturals $\mathcal{N}$. Throughout the remainder of this section, we assume a fixed domain $D = (\Pi, \Sigma, \Phi)$.

A *labeling* W of a set of variables $A \subseteq \Pi$ associates each variable $a \in A$ with a Σ_a valued Φ_a-trace denoted W_a. We denote the set of all such labelings of A by A^D. As an example, let A consist of the variables "in" and "out." Let $\Sigma_{\text{in}} = \mathcal{R}$ and $\Phi_{\text{in}} = \mathcal{R}$, and let $\Sigma_{\text{out}} = \mathcal{N}$ and $\Phi_{\text{out}} = \mathcal{N}$. A labeling W of A might associate the $\mathcal{R}$-trace $W_{\text{in}} = \lambda x \in \mathcal{R}[0, 3.5) \cdot x^2$ with "in" and the $\mathcal{N}$-trace $W_{\text{out}} = \langle 3, 1, 15 \rangle$ with "out."

The notation for trace concatenation extends naturally to labelings. Let A be a set of variables. Let $U \in A^D$ and $V \in A^D$ be labelings of A. Then $UV \in A^D$ is a labeling that associates each variable $a \in A$, with the trace concatenation $U_a V_a$.

$$(UV)_a \stackrel{\text{def}}{=} U_a V_a$$

We denote the unique labeling of a variable-set A that associates each variable $a \in A$ with the empty-trace by ϵ_A. For a fixed variable-set A, it is straightforward to show that the concatenation operator is associative with identity ϵ_A, and that the set A^D is closed under finite concatenations; hence, the set A^D and concatenation form a monoid. However, not every property of trace concatenation holds for labelings. For example, the identity $uv = vu$ holds for traces if and only if $u = v$ or one of u or v is the empty-trace ϵ. The same is not true for labelings. Let u and v be non-empty traces in $\Sigma_a^{\Phi_a}$ and $\Sigma_b^{\Phi_b}$ respectively. Let U and V be labelings of $\{a, b\}$ such that $U_a = u$, $U_b = \epsilon$, $V_a = \epsilon$ and $V_b = v$. Neither U nor V are empty labelings, yet $UV = (U_a V_a, U_b V_b) = (u, v) = (V_a U_a, V_b, U_b) = VU$.

If W is a labeling of A, and $B \subseteq A$, we denote by $W|_B$ the labeling of B that associates each variable $b \in B$ with W_b. It should be clear for any such variable-sets A and B, the restriction $|_B$ defines a homomorphism from A^D to B^D with respect to trace concatenation. That is, if U and V are both labelings of A and $B \subseteq A$, then $(UV)|_B = U|_B V|_B$.

A *trace-automaton* is a quadruple, $m = (A(m), S(m), I(m), \Delta(m))$, in which $A(m) \subseteq \Pi$ is a set of variables, $S(m)$ is a set of states, $I(m) \subseteq S(m)$ is a set of initial states, and $\Delta(m) \subseteq A(m)^D \times S(m) \times S(m)$ is a transition relation. A *chain of m of length z* is a

Table 1: A 1-bit Counter

$m_{c1} \in \mathcal{M}_{(\Pi,\Sigma,\Phi)}$

$a \in \Pi$	Σ_a	Φ_a
q	$\{0,1\}$	$\mathcal{N}$
r	$\{0,1\}$	$\mathcal{N}$

$A(m_{c1}) = \{q, r\}$

$S(m_{c1}) = \{0,1\}$

$I(m_{c1}) = \{0\}$

$(W, s, s') \in \Delta(m_{c1}) \stackrel{\text{def}}{\equiv}$		
(W_q, W_r)	s	s'
$(\langle 0 \rangle, \langle 0 \rangle)$	0	0
$(\langle 1 \rangle, \langle 0 \rangle)$	0	1
$(\langle 0 \rangle, \langle 1 \rangle)$	1	1
$(\langle 1 \rangle, \langle 1 \rangle)$	1	0

sequence of states $s_0, s_1, ..., s_z \in S(m)$, and a sequence of labelings $w_1, w_2, ..., w_z$, of $A(m)$, such that $s_0 \in I(m)$, and such that $(w_i, s_{i-1}, s_i) \in \Delta(m)$ for each $1 \leq i \leq z$. The following notation will be used to depict such a chain.

$$s_0 \xrightarrow{w_1} s_1 \xrightarrow{w_2} \cdots \xrightarrow{w_z} s_z$$

We say that the labeling W is in $L(m)$, the *language accepted by m,* if and only if W is the concatenation of the labelings $w_1, w_2, ..., w_z$ for such a chain. We denote by $\mathcal{M}_D$, the set of all trace-automata over the domain D.

Example 1: A 1-bit binary counter

To illustrate the features of trace-automata, we give several examples. Table 1, which defines a discrete 1-bit binary counter, is an example of a standard tabular notation that we have adopted. The table begins with a title that names the automaton being described, in this case m_{c1}, and names the components of the domain to which it belongs. The box that follows is a partial description of this domain. It gives the value and time-domains associated with the labels that this automaton will use. For m_{c1}, Σ associates the value-domain $\{0,1\}$ and Φ associates the time-domain $\mathcal{N}$ with the two variables "q" and "r."

The second box enumerates the variable-set A. This identifies the input tapes that the automaton will read, generally the same variables whose domains were given in the preceding box. The third box gives the state-set S. The automaton m_{c1} has two states, 0 and 1. The fourth box gives the initial state-set I, in this case the singleton $\{0\}$. Finally the state-transition relation is given, either as a logical expression or, as in this case, in a tabular form.

The trace-automata formalism does not explicitly formalize the notion of "output." Instead, the question of which tapes are considered to be "outputs" and which are considered to be "inputs" is left as a matter of interpretation. In the preceding example, suppose that we consider the variable q to be an "input," and the variable r to be an "output." Viewed in this way, we can see that the automata counts (mod 2) the number of ones

Table 2: An Edge Detector

$m_{\text{edge}} \in \mathcal{M}_{(\Pi,\Sigma,\Phi)}$

$a \in \Pi$	Σ_a	Φ_a
p	$\{0,1\}$	$\mathcal{N}$
q	$\{0,1\}$	$\mathcal{N}$

$A(m_{\text{edge}}) = \{\mathrm{p}, \mathrm{q}\}$

$S(m_{\text{edge}}) = \{0,1\}$

$I(m_{\text{edge}}) = \{0\}$

$(W, s, s') \in \Delta(m_{\text{edge}}) \stackrel{\text{def}}{\equiv}$

(W_p, W_q)	s	s'
$(\langle 0\rangle, \langle 0\rangle)$	0	0
$(\langle 1\rangle, \langle 1\rangle)$	0	1
$(\langle 0\rangle, \langle 0\rangle)$	1	0
$(\langle 1\rangle, \langle 0\rangle)$	1	1

that appear on its "input." Notice also, that when viewed this way, the automata is a Moore machine[9]: The next state is a function of the current state and input symbol; the "output" symbol produced is a function of the current state.

Example 2: An Edge Detector

The second example can be interpreted as an edge-detector. Consider the variable p to represent the "input" and the variable q to represent the "output." As the machine operates, the output "q" takes on the value 1 each time the input "p" takes on the value 1 after first taking on the value 0. Notice that under this interpretation, the automata is a Mealy machine[8]: The next state and the "output" symbol are both functions of the current state and the input symbol.

Example 3: An Integrator

The first two examples were discrete; all of the tapes were $\mathcal{N}$-traces. In contrast the third example is a continuous integrator; all of its tapes are $\mathcal{R}$-traces. It accepts a trace W if and only if $W_{\dot{x}}$ and W_x are the same length, and if the trace W_x could be obtained by integrating the trace $W_{\dot{x}}$. Although we view the machine as an integrator, and hence view $\dot{x}$ as its "input" and x as its "output," we could equally well see the machine as a differentiator, with "input" x and "output" $\dot{x}$. The formalism itself imposes no notion of direction or causality.

Example 4: A Switch

The fourth example is a *hybrid* model; some of its inputs are $\mathcal{R}$-traces, while some – in this case one – are $\mathcal{N}$-traces. The model describes an analog switch. To understand the model, the reader should view the parameters i_0 and i_1 and c as "inputs," while viewing the parameter o as an "output." On each transition, the switch consumes equal-length chunks from its two continuous "inputs" i_0 and i_1, and one symbol from its discrete "input" c. If the symbol 0 is consumed from c, then the trace consumed from o must equal that consumed from i_0. Conversely, if the symbol 1 is consumed, then the trace consumed from o must equal that consumed from i_1.

Table 3: An Integrator

$m_{\text{int}} \in \mathcal{M}_{(\Pi,\Sigma,\Phi)}$

$a \in \Pi$	Σ_a	Φ_a
$\dot{x}$	$\mathcal{R}$	$\mathcal{R}$
x	$\mathcal{R}$	$\mathcal{R}$

$A(m_{\text{int}}) = \{\dot{x}, x\}$

$S(m_{\text{int}}) = \mathcal{R}$

$I(m_{\text{int}}) = \mathcal{R}$

$$
\begin{aligned}
&(W, s, s') \in \Delta(m_{\text{int}}) \stackrel{\text{def}}{\equiv} \\
&\qquad |W_{\dot{x}}| = |W_x| \wedge \\
&\qquad \forall r \in [0, |W_{\dot{x}}|) \cdot W_x(r) = s + \textstyle\int_0^r W_{\dot{x}}(t)dt \wedge \\
&\qquad s' = s + \textstyle\int_0^{|W_{\dot{x}}|} W_{\dot{x}}(t)dt
\end{aligned}
$$

Table 4: An Analogue Switch

$m_{\text{switch}} \in \mathcal{M}_{(\Pi,\Sigma,\Phi)}$

$a \in \Pi$	Σ_a	Φ_a
i_0	$\mathcal{R}$	$\mathcal{R}$
i_1	$\mathcal{R}$	$\mathcal{R}$
o	$\mathcal{R}$	$\mathcal{R}$
c	$\{0, 1\}$	$\mathcal{N}$

$A(m_{\text{switch}}) = \{i_0, i_1, o, c\}$

$S(m_{\text{switch}}) = \{0\}$

$I(m_{\text{switch}}) = \{0\}$

$$
\begin{aligned}
&(W, s, s') \in \Delta(m_{\text{switch}}) \stackrel{\text{def}}{\equiv} \\
&\qquad |W_{i_0}| = |W_{i_1}| \wedge \\
&\qquad (W_c = \langle 0 \rangle \ \wedge \ W_o = W_{i0}) \ \vee \ (W_c = \langle 1 \rangle \ \wedge \ W_o = W_{i1})
\end{aligned}
$$

Table 5: A transliteration

$m_= \in \mathcal{M}_{(\Pi,\Sigma,\Phi)}$

$a \in \Pi$	Σ_a	Φ_a
a	$\mathcal{R}$	$\mathcal{R}$
b	$\mathcal{R}$	$\mathcal{R}$
$A(m_=) = \{a, b\}$		
$S(m_=) = \{0\}$		
$I(m_=) = \{0\}$		
$(W, s, s') \in \Delta(m_{\text{switch}}) \stackrel{\text{def}}{\equiv} W_a = W_b$		

Example 5: A Transliteration

Finally, we give an example of a class of machines called *transliterations*. A transliteration is an essentially stateless machine that enforces a point-wise relationship between its inputs traces. Its behaviour does not depend in any way on its history. For example, the machine $m_=$, presented in Table 5, accepts any labeling W of $\{a, b\}$ satisfying $|W_a| = |W_b|$ and $W_a(t) = W_b(t)$ for all $t \in \mathcal{R}[0, |W_a|)$.

We build large models, by composing smaller components. The following four equations define the composition $m_1 \| m_2$ for any trace-automata m_1 and m_2 from the same domain.

$$
\begin{aligned}
A(m_1 \| m_2) &\stackrel{\text{def}}{=} A(m_1) \cup A(m_2) \\
S(m_1 \| m_2) &\stackrel{\text{def}}{=} S(m_1) \times S(m_2) \\
I(m_1 \| m_2) &\stackrel{\text{def}}{=} I(m_1) \times I(m_2) \\
(W, (s_1, s_2), (t_1, t_2)) \in \Delta(m_1 \| m_2) &\stackrel{\text{def}}{\equiv} \\
& (W|_{A(m_1)}, s_1, t_1) \in \Delta(m_1) \;\wedge\; (W|_{A(m_2)}, s_2, t_2) \in \Delta(m_2)
\end{aligned}
$$

We extend the restriction notation from labelings to automata in the obvious way. If m is an automaton, and $B \subseteq A(m)$, we use the notation $m|_B$ to denote the automaton defined as follows:

$$
\begin{aligned}
A(m|_B) &\stackrel{\text{def}}{=} B \\
S(m|_B) &\stackrel{\text{def}}{=} S(m) \\
I(m|_B) &\stackrel{\text{def}}{=} I(m) \\
(W, s, t) \in \Delta(m|_B) &\stackrel{\text{def}}{\equiv} \exists W' \in A^D \cdot (W'|_B = W) \;\wedge\; (W', s, t) \in \Delta(m)
\end{aligned}
$$

Proposition 3.1 *Suppose m is a trace-automaton and $B \subseteq A(m)$. For any labeling W of $A(m)$, if $W \in L(m)$, then $W|_B \in L(m|_B)$.*

We could use the tabular notation from the preceding examples to describe such a composite automata. For complicated machines, however, this notation becomes hard to understand. Instead, we shall rely on illustrations that show the structure of a composite machine. For example, Figure 1 shows the composition of m_{c1} and m_{edge} from Examples 1

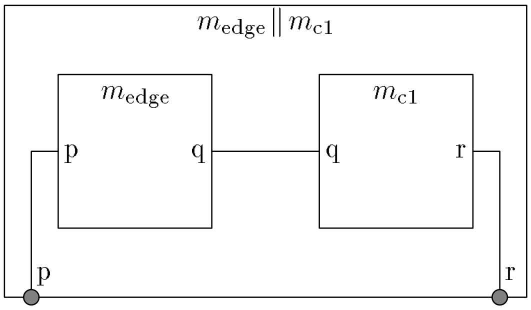

Figure 1: Composition of two trace automata.

and 2 respectively. The machines are represented as boxes with attachment points at the perimeter for each of their variables. Component machines are drawn nested inside the box representing their composition. Lines are drawn to indicate "connections" between components. The obvious renaming — unnecessary in this example — is presumed to take place, so that variables that are connected by such a line share a unique name. Domain restriction is indicated by the absence of a line connecting the component variables to the perimeter of the box representing the composition. In this example, restriction to the set $\{p, r\}$ is shown.

To illustrate the use of this formalism, we shall construct a model of a simple hybrid system. The system models two trains that move in the same direction along a continuous loop of track. Each train is connected to a central controller. At regular intervals, the controller measures the position of each train. A train which is too close to the other, where distance is measured in the direction of travel, is instructed to stop. If it is not too close to the other, it is instructed to proceed forwards.

The motion of each train can be described using differential equations. If the most recent control action was "go," then the velocity will be governed by Equation 1 below in which v_{max} and k_{go} are fixed positive constants.

$$\dot{v} = k_{\text{go}}(v_{\text{max}} - v) \tag{1}$$

If the most recent control action was "stop" then the brakes are applied and the velocity reduces at a constant rate k_{stop} until it reaches zero.

$$\dot{v} = \begin{cases} -k_{\text{stop}} & \text{if } v > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

We can use trace-automata to model these equations as illustrated in Figure 2. We begin by viewing v and $\dot{v}$ as independent variables, and define the transliteration corresponding to each equation. For example, we define the transliteration m_{v1} that accepts labelings W such that W labels v and $\dot{v}$ with equal-length traces satisfying $W_{\dot{v}}(t) = k_{\text{go}}(v_{\text{max}} - W_v(t))$ for all $t \in [0, |W_v|)$. Similarly, we define the transliteration m_{v2} that accepts labelings W satisfying Equation 2. The two transliterations represent the response of the trains to the two different control inputs. We can model this dependency by composing the transliterations with the switch presented in example 4.

So far, our model treats the variables v and $\dot{v}$ as if they were independent. Of course the original equations are only satisfied when $\dot{v}$ is the derivative of v. This requirement is captured by composing the model with the integrator from example 3. Ultimately, the position p of each train is the integral (modulo the length L of the track) of the train's velocity. Thus, to complete our model, we compose the automata obtained so far, with a second integrator. This one is a slight variation on the integrator of Example 3 in that it integrates modulo the track-length L. Since we are ultimately interested only in the train's control-input and its position, we restrict the final model to the variable-set $\{c, p\}$.

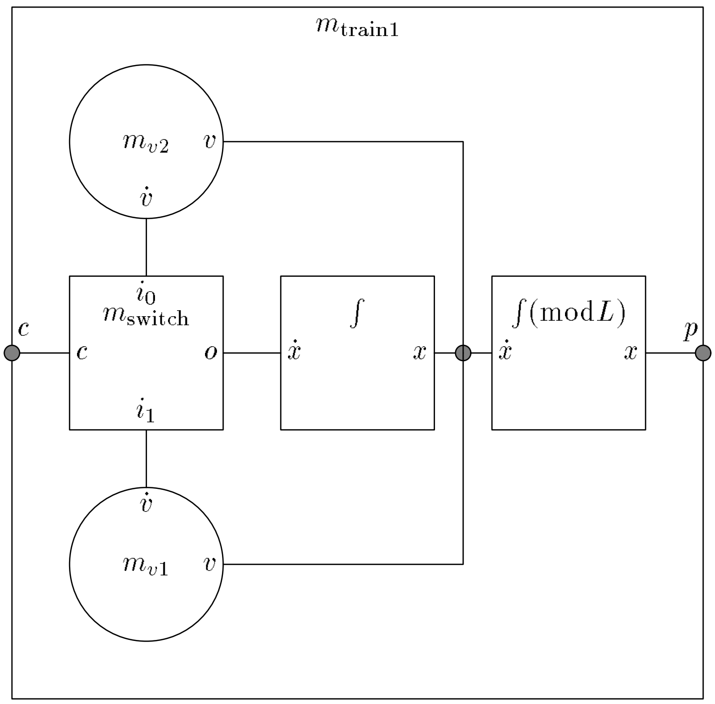

Figure 2: The dynamics of a train.

Having thus modeled the dynamics of a train, we turn our attention towards the controller. The controller is able to measure the position of each train, but without much precision. In fact, the best the controller can do is to determine which of 8 equal-length track sections the train is in at the time the measurement is taken. It alternates between measuring the positions of the two trains in this way, and sending an instruction, either `GO` or `STOP`, to each one. The instructions are sent simultaneously at a regular fixed interval. The measurements occur at some indeterminate time between successive instructions.

We use a discrete trace-automaton to model the control algorithm. The algorithm maintains a single bit of state information for each train, which can have either of the values `GO` or `STOP`. The state resulting from each transition is based on the sampled positions of the trains. If a train is within one track section of the train in front of it, the resulting state-bit will be `STOP`. If the train is not too close, the resulting bit will be `GO`. The controller output during each transition is the value of the corresponding state bit at the start of the transition. In this way, the result of sampling in one time unit, is reflected in the control action taken during the next. The position measurement process is modeled by a pair of hybrid automata, one for each train, as given in Table 6. The entire system is depicted in Figure 3.

Table 6: The sampling process

$m_{\text{sample}} \in \mathcal{M}_{(\Pi,\Sigma,\Phi)}$

$a \in \Pi$	Σ_a	Φ_a
x	$\mathcal{R}[0, L)$	$\mathcal{R}$
y	$\mathcal{N}[0, 7]$	$\mathcal{N}$
$A(m_{\text{sample}}) = \{\text{x}, \text{y}\}$		
$S(m_{\text{sample}}) = \{0\}$		
$I(m_{\text{sample}}) = \{0\}$		
$(W, s, s') \in \Delta(m_{\text{sample}}) \stackrel{\text{def}}{\equiv}$ $\lvert W_x\rvert = 1 \ \wedge\ \lvert W_y\rvert = 1 \ \wedge\ \exists r \in [0, 1) \cdot W_y(0) = \lfloor W_x(r) * 8.0/L \rfloor$		

One of the main features of a hybrid system is that different models of time are appropriate for different components. Indeed, reconciling these differences is one problem that makes hybrid systems particularly challenging. The trace-automata formalism does not attempt to dictate the terms of such a reconciliation. There is nothing in the formalism that requires the same rate of consumption of traces attached to different variables. The temporal relationship between happenings of different variables is specified, if at all, by the automata themselves. For example, consider the "switch" machine of Example 4. The switch model requires that the $\mathcal{R}$-traces attached to i_1, i_2 and o are consumed at the same rate. However, it establishes no relation between this rate, and the rate at which the discrete control input c is consumed. The idea is that the switch accepts control signals at whatever rate the controller produces them. In our model, this rate is actually specified by the "sampler" machine in table 6. The sampler "produces" one control token for each unit-length chunk of its continuous input traces.

4 Conservative Approximation

We have constructed a model that represents the ways in which the system is capable of behaving. Ultimately, we shall want to verify that these behaviours are the ones that we want. A verification domain, $V = (\mathcal{M}, \mathcal{Q}, \models)$, is a triple in which $\mathcal{M}$ is a set of models, $\mathcal{Q}$ is a set of specifications, and $\models \subseteq \mathcal{M} \times \mathcal{Q}$ is a binary satisfaction relation. When $m \models q$ holds, we say that the model m *satisfies* the specification q.

While it may be easy to *define* the satisfaction relation $\models$, it will often be quite intractable to decide whether it holds for a specific model and specification. One way to combat this is to translate the question and the model into a different, more tractable framework. Let $\widehat{V} = (\widehat{\mathcal{M}}, \widehat{\mathcal{Q}}, \widehat{\models})$ be such a framework. Let $\Psi : \mathcal{Q} \mapsto \widehat{\mathcal{Q}}$ be an *abstraction* function that translates specifications from the complex framework V to the alternate framework $\widehat{V}$.

Suppose we now wish to compare a model $m \in \mathcal{M}$ to an *approximation* $\widehat{m} \in \widehat{\mathcal{M}}$. Let $G_\Psi(m, \widehat{m}) \subseteq Q$ denote the set of specifications on which m and $\widehat{m}$ agree.

$$G_\Psi(m, \widehat{m}) \stackrel{\text{def}}{\equiv} \left\{ q \in Q \;\middle|\; \widehat{m} \mathrel{\widehat{\models}} \Psi(q) \equiv m \models q \right\}$$

That is, a specification q is in $G_\Psi(m, \widehat{m})$ if and only if both q and $\Psi(q)$, or neither q nor

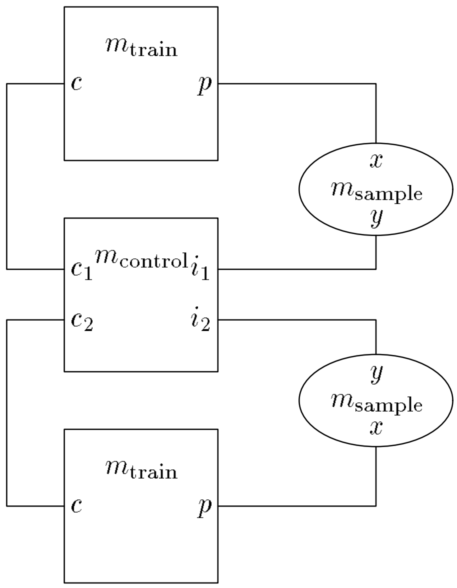

Figure 3: The train-system model.

$\Psi(q)$, are satisfied by m and $\widehat{m}$ respectively. We can compare the quality of approximations $\widehat{m}_1$ and $\widehat{m}_2$ by comparing their agreement sets. Let $\widehat{m}_1$ and $\widehat{m}_2$ be approximations to m with respect to Ψ. We say that $\widehat{m}_2$ is *as good an approximation* of m as $\widehat{m}_1$ if its agreement set contains the agreement set of $\widehat{m}_1$.

$$\widehat{m}_1 \preceq_\Psi \widehat{m}_2 \stackrel{\text{def}}{\equiv} G_\Psi(m, \widehat{m}_1) \subseteq G_\Psi(m, \widehat{m}_2)$$

We say a model $\widehat{m}$ is a *conservative* approximation of another m with respect to a translator Ψ if and only if for every specification $\Psi(q)$ that is satisfied by $\widehat{m}$, the specification q is satisfied by m. That is, for every $q \in Q$ we must have

$$\widehat{m} \mathrel{\widehat{\models}} \Psi(q) \Longrightarrow m \models q$$

The machine $\widehat{m}$ is conservative because if it satisfies a specification $\widehat{F}$, then every related specification $q \in \Psi^{-1}(\widehat{F})$ must be satisfied by m. It is an approximation, because it need not satisfy every specification $\Psi(q)$ for which m satisfies q.

5 Approximations Between Automata

In the previous section, we gave a very general definition of conservative approximation with respect to an abstraction function Ψ. Here, we present a practical method for constructing such a function for trace-automata specifications. Before doing so, however, we develop a verification domain $(\mathcal{M}_D, \mathcal{Q}_D, \models_D)$ for each trace-automata domain D.

Let $D = (\Pi, \Sigma, \Psi)$ be a trace automata domain. Let $\mathcal{M}_D$ be the set of all trace automata over D. Recall from Section 3 that Π^D represents the set of all labelings of Π in D. We define $\mathcal{Q}_D$, the set of *specifications* over the domain D, as $\mathcal{Q}_D \stackrel{\text{def}}{\equiv} 2^{\Pi^D}$, the power set of Π^D. Each specification $F \in Q_D$ is a set of *prohibited* labelings of Π. We think of F as a "failure set," in that a verification should fail if the model admits any behaviour in this set. Thus, the satisfaction relation $\models_D$ is defined as follows:

$$m \models_D F \stackrel{\text{def}}{\equiv} L(m) \cap \left(F|_{A(m)}\right) = \emptyset$$

Thus, we have defined a verification framework for the trace-automata domain D. The models are the trace automata over D. The specifications are sets of labelings of Π that represent prohibited evolutions. A model m satisfies a specification F if it does not permit any behaviour in the failure set.

5.1 Abstracting Specifications

Let $D = (\Pi, \Sigma, \Phi)$ and $\widehat{D} = (\Pi, \widehat{\Sigma}, \widehat{\Phi})$ be two trace-automata domains, that share the same variable-set Π. Suppose that for each variable $a \in \Pi$, we have defined an abstraction function $\psi_a : \Sigma_a^{\Phi_a} \mapsto \widehat{\Sigma}_a^{\widehat{\Phi}_a}$ that maps traces of the variable a in the original domain D to traces of the same variable in the abstract domain $\widehat{D}$. Let $A \subseteq \Pi$ be any set of variables. We combine the functions ψ_a, which map from traces to traces, to produce the abstraction function $\psi_A : A^D \mapsto A^{\widehat{D}}$ from labelings in A^D to labelings in $A^{\widehat{D}}$ in the obvious way, so that $(\psi_A(W))_a \stackrel{\text{def}}{=} \psi_a(W_a)$ for each $a \in A$. When the set A is clear from the context, as it almost always will be, we omit the subscript A in ψ_A and simply refer to ψ. A function built in this way is a *separable* abstraction function. It is separable, in that the traces of individual variables are abstracted independently. The following property is an obvious consequence of the way in which ψ is defined by combining independent components ψ_a.

Proposition 5.1 *If A and B are sets of variables, such that $B \subseteq A$, then the following identity holds for all separable abstraction functions ψ and all labelings W of A.*

$$\psi(W)|_B = \psi(W|_B)$$

Recall that a trace-automata specification for the domain D is a set of labelings in Π^D. Thus, by extending the abstraction function ψ from labelings to sets of labelings, we can define an abstraction function mapping from specifications in $\mathcal{Q}_D$ to specifications in $\mathcal{Q}_{\widehat{D}}$. We define the operator Γ which performs such an extension:

$$(\Gamma\psi)(F) = \{\psi(W) \mid W \in F\}$$

Throughout the remainder of this paper, we will build conservative approximations based on two particular abstraction functions that map from real valued $\mathcal{R}$-traces to $\mathcal{N}$-traces of integer pairs. Let w be an arbitrary real-valued $\mathcal{R}$-trace. If w is empty, then it is mapped onto the empty-trace. If w is non-empty, we partition it into a sequence of zero or more traces of length 1, and one non-empty trace of length one or less. Let $w_0, w_1, ..., w_k$ be this partitioning, so that $w = w_0 w_1 \cdots w_k$, $|w_i| = 1$ for each $i < k$, and $0 < |w_k| \leq 1$. Each sub-trace w_j is mapped to a pair of integers (l_j, u_j) such that the interval $\mathcal{R}[l_j, u_j]$ is the smallest interval with integer bounds that contains the range of w_j. That is, we must have $l_j \leq w_j(t) \leq u_j$ for all $t \in \mathcal{R}[0, |w_j|)$ and for all $0 \leq j \leq k$. In this way, we map the $\mathcal{R}$-trace w to a $\mathcal{N}$-trace of integer-pairs, $\langle \widehat{w}(1), \widehat{w}(2), ..., \widehat{w}(k)\rangle$ such $\widehat{w}(j) = (l_j, u_j)$ for each $j \in 0, 1, ..., k$ as illustrated in Figure 4.

This mapping is suitable for all the $\mathcal{R}$-traces in the model with the exception of those representing the position of a train. Recall that trains run on a circular track of length L

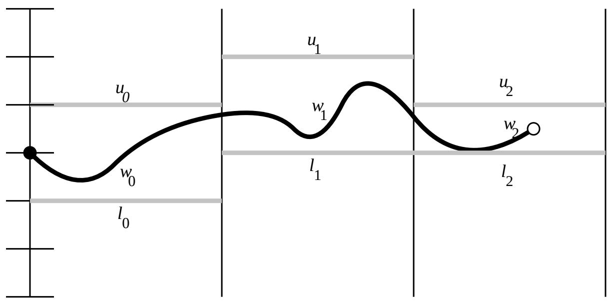

Figure 4: Abstracting a real-trace by a sequence if integer-pairs

which wraps around so that 0 and L represent the same physical location. For simplicity we assume that L is an integer. Let x be any real number, greater than 0 but less than $L/2$. Our discrete representation of small traces needs to distinguish between the relatively small motion from $L-x$ to $0+x$, passing through 0 but not through $L/2$, and the relatively large trip backwards from $L-x$ to $0+x$ passing through $L/2$ but not through 0. However, the function as described so far represents both of these traces by the pair $(0, L)$.

For this reason we will abstract traces representing the position of a train in a slightly different way. Let S be the closure of the range of w. If there exists real numbers $0 < a < b < L$ such that S is the union of the intervals $[0, a]$ and $[b, L]$, then we represent w by the integer pair $(\lceil b \rceil, \lfloor a \rfloor)$. Note that $\lceil b \rceil > \lfloor a \rfloor$. On the other hand, if no such numbers exist then we revert to the established representation (l, u) where l and u are respectively the greatest and the least integers such that S is contained in the interval $[l, u]$.

For example let $x = \frac{1}{2}$. Let w_j be a short trace representing the forward motion of a train from $0 + x$ to $L - x$. The trace would be abstracted by the pair $(0, L)$. On the other hand, let w_j represent the forward motion of the train from $L - x$ to x. In this case there exists $a = x$ and $b = L - x$ so that the range of w is the union of $[0, a]$ and $[b, L]$. Thus, we abstract the trace w_j by the pair $(L, 0)$.

Let $\sigma : \Pi^D \mapsto \Pi^{\widehat{D}}$ be the separable abstraction function from labelings in D to labelings in $\widehat{D}$ that uses the functions defined above to abstract the individual traces. We can build the corresponding abstraction function for specifications using the operator (Γ). The result, $(\Gamma\sigma)$ translates specifications in the domain D into specifications in the abstract domain $\widehat{D}$.

Having constructed a separable abstraction function, we seek to construct an approximation of the system that is conservative with respect to it. The following theorem identifies the conditions under which a trace-automata $\widehat{m}$ is a conservative approximation of another m with respect to any separable abstraction function $(\Gamma\psi)$.

Theorem 5.2 *The automaton $\widehat{m}$ is a conservative approximation of m under any separable abstraction $(\Gamma\psi)$ if and only if $\psi(W) \in L(\widehat{m})$ for every $W \in L(m)$.*

Proof: To prove the "if" part assume that $\psi(W) \in L(\widehat{m})$ for all $W \in L(m)$. We must show that for any specification $F \in \mathcal{Q}_D$, if $\widehat{m}$ satisfies $(\Gamma\psi)(F)$, then m satisfies F. Suppose, with an eye towards contradiction, that F is a specification that m does not satisfy, yet $\widehat{m}$ satisfies $(\Gamma\psi)(F)$. Then there must be a labeling V of Π such that $V|_A \in F|_A \cap L(m)$.

Let V be such a labeling. By assumption, $\psi(V|_A)$; hence, by proposition 5.1, $\psi(V)|_A$ is in $L(\widehat{m})$. Since $V \in F$, we must have $\psi(V) \in (\Gamma\psi)(F)$. Hence $\psi(V)|_A \in (\Gamma\psi)(F)|_A \cap L(\widehat{m})$. But this intersection is supposed to be empty since $\widehat{m}$ is supposed to satisfy $(\Gamma\psi)(F)$. Thus the sought contradiction is established, and we conclude $\left(\widehat{m} \models_{\widehat{D}} (\Gamma\psi)(F)\right) \Longrightarrow (m \models_D F)$.

To prove the "only if" part, assume that $\widehat{m}$ is a conservative approximation of m under $(\Gamma\psi)$. Let W be an arbitrary labeling in $L(m)$. We must show that $\psi(W) \in L(\widehat{m})$. Let F be any specification such that $q|_A = \{W\}$. For example let $F = \{V\}$ where V is the labeling of Π such that $V_a = W_a$ when $a \in A$, and $V_a = \epsilon$ otherwise. Clearly m does not satisfy F. Since $\widehat{m}$ is a conservative approximation of m under $(\Gamma\psi)$, $\widehat{m}$ cannot satisfy $(\Gamma\psi)(F)$, hence $\psi(V) \in L(\widehat{m})$ as required. □

5.2 Approximating Integration

To illustrate the ideas in this section we develop a sequence of progressively better approximations of the integrator from Section 3. Let $\widehat{m}$ be a trace-automata with integer valued states: $S(m) = \mathcal{Z}$. Whereas the original integrator accepted labelings associating the variables x and $\dot{x}$ with real-valued $\mathcal{R}$-traces, the approximation will accept labelings associating x and $\dot{x}$ with $\mathcal{N}$-traces of integer pairs. Let $\Delta(m)$ be the universal state-transition relation: $\Delta(m) = A(m)^{\widehat{D}} \times S(m) \times S(m)$. Clearly, with this transition relation, $\widehat{m}$ will accept any labeling, $L(\widehat{m}) = A(\widehat{m})^{\widehat{D}}$. The model $\widehat{m}$ is a conservative approximation of the integrator m, but it is the poorest conservative approximation possible, and is not very useful.

To improve the approximation we make the transition relation smaller. For a start, we only allow $\widehat{m}$ to accept input symbols one at a time.

$$(\widehat{W}, \widehat{s}, \widehat{s'}) \in \Delta(\widehat{m}) \Longrightarrow \left|\widehat{W}_{\dot{x}}\right| = \left|\widehat{W}_x\right| = 1$$

This constrains the language $L(\widehat{m})$ so that it only contains labelings that associate x and $\dot{x}$ with traces of the same length. The machine is still a conservative approximation of the integrator, since the integrator also accepted only labelings associating x and $\dot{x}$ with traces of equal length, and the abstraction function σ preserves this property.

Let W be an arbitrary labeling of $\{x, \dot{x}\}$. Observe that if the integrator m accepts W it can do so by means of a chain

$$s_0 \xrightarrow{W_1} s_1 \xrightarrow{W_2} \cdots \xrightarrow{W_k} s_k$$

such that W_k associates x and $\dot{x}$ with non-empty traces of length one or less, and each W_i, for $i < k$ associates x and $\dot{x}$ with traces of length exactly 1. That is, W_k matches the partitioning used by the abstraction function ψ.

Recall that the abstraction function maps each of these sub-traces to a pair of integers. If $\widehat{m}$ is to be a conservative approximation of m, we must be sure that it has a chain

$$\widehat{s}_0 \xrightarrow{\widehat{W}_1} \widehat{s}_1 \xrightarrow{\widehat{W}_2} \cdots \xrightarrow{\widehat{W}_k} \widehat{s}_k$$

where $\widehat{W}_j = \psi(W_j)$ for each $1 \leq j \leq k$. To do this, we define a state abstraction function ρ from $S(m)$ to $S(\widehat{m})$, and ensure the particular chain in which $\widehat{s}_j = \rho(s_j)$ is a chain of $\widehat{m}$.

Define $\rho(x) = \left\lfloor x + \frac{1}{2} \right\rfloor$, so that intervals from $x - \frac{1}{2}$ to $x + \frac{1}{2}$ are mapped onto each integer x. We let the initial state-set $I(\widehat{m})$ be the image of $I(m)$ under ρ. Let (W, s, s') be any transition in the chain of m. Let $(\dot{l}, \dot{u})$ and (l, u) be the single elements of the "input" trace $\widehat{W_{\dot{x}}}$ and the "output" trace $\widehat{W_x}$ respectively. At the beginning of the transition, the integral W_x is s. At the end, the integral W_x is s'. However, knowing the values of W_x at only these points does not necessarily constrain the intermediate values and hence does not constrain l and u.

Suppose, however, that $\dot{l} \geq 0$. Then the integrand $W_{\dot{x}}$, which it abstracts, is everywhere non-negative. In this case, the integral W_x must increase monotonically, starting with a minimum value of s and ending with a maximum value of s'. Thus, we can constrain u and l so that $u = \lceil s' \rceil$ and $l = \lfloor s \rfloor$. Similarly, if $\dot{u} \leq 0$, then $u = \lceil s \rceil$ and $l = \lfloor s' \rfloor$. A little algebra allows us to establish the following conditions.

$$\left.\begin{array}{lcccl} \rho(s') & \leq & u & \leq & \rho(s')+1 \\ \rho(s)-1 & \leq & l & \leq & \rho(s) \end{array}\right. \quad \text{when } \dot{l} \geq 0$$

$$\left.\begin{array}{lcccl} \rho(s) & \leq & u & \leq & \rho(s)+1 \\ \rho(s')-1 & \leq & l & \leq & \rho(s') \end{array}\right. \quad \text{when } \dot{u} \leq 0$$

If $\dot{l} < 0$ and $\dot{u} > 0$, then the integrand is sometimes greater and sometimes less than zero so integral will not be monotonic. In this case tight bounds are somewhat more complicated to derive. It is easily to see, however, that the maximum value of W_x cannot exceed the sum of the starting value s and the maximum value of $W_{\dot{x}}$. Less obvious perhaps is that it cannot exceed the ending value s' less the minimum value of $W_{\dot{x}}$. To see this suppose that it achieves a value x at some time t such that $x > s' - \min(W_{\dot{x}})$. Then to go from x to s' would be impossible, since it would require an rate of change less that $\min(W_{\dot{x}})$. Similarly, the minimum value of W_x must be greater than $s + \min(W_{\dot{x}})$ and $s' - \max(W_{\dot{x}})$. Of course, the maximum value of W_x must be as great as s and s', both of which must be less than its minimum value. After some manipulation we can conclude that the following inequalities hold.

$$\begin{array}{lcccl} \rho(s) & \leq & u & \leq & \rho(s)+\dot{u}+1 \\ \rho(s') & \leq & u & \leq & \rho(s')-\dot{l}+1 \\ \rho(s)+\dot{l}-1 & \leq & l & \leq & \rho(s) \\ \rho(s')-\dot{u}-1 & \leq & l & \leq & \rho(s') \end{array}$$

Thus we can safely restrict the trace $\widehat{W_x}$ associated with a transition from $\widehat{s}$ to $\widehat{s'}$ in the following ways.

$$\begin{array}{lcccl} \widehat{s'} & \leq & u & \leq & \widehat{s'}+1 \\ \widehat{s}-1 & \leq & l & \leq & \widehat{s} \end{array} \quad \text{when } \dot{l} \geq 0$$

$$\begin{array}{lcccl} \widehat{s} & \leq & u & \leq & \widehat{s}+1 \\ \widehat{s'}-1 & \leq & l & \leq & \widehat{s'} \end{array} \quad \text{when } \dot{u} \leq 0$$

$$\begin{array}{lcccl} \widehat{s} & \leq & u & \leq & \widehat{s}+\dot{u}+1 \\ \widehat{s'} & \leq & u & \leq & \widehat{s'}-\dot{l}+1 \\ \widehat{s}+\dot{l}-1 & \leq & l & \leq & \widehat{s} \\ \widehat{s'}-\dot{u}-1 & \leq & l & \leq & \widehat{s'} \end{array} \quad \text{otherwise}$$

Using similar reasoning, we can also restrict the "input" traces $\widehat{W_{\dot{x}}}$ as follows:

$$\begin{array}{lcccl} \widehat{s'}-\widehat{s} & \leq & \dot{u} & \text{if } \widehat{s'} \leq \widehat{s} \\ \widehat{s'}-\widehat{s} & \geq & \dot{l} & \text{if } \widehat{s'} \leq \widehat{s} \end{array}$$

The result is still a conservative approximation of the continuous integrator. If m has a transition (W, s, s'), then $\widehat{m}$ still has a corresponding transition $(\sigma(W), \rho(s), \rho(s'))$.

Note that the model is still too conservative to show that speed of a train changes in response to acceleration. To see this, observe that a transition from any state $\widehat{s}$ to itself is permitted for any input $(\dot{l}, \dot{u})$. Thus, if the integrator input represents acceleration, and the output represents velocity, the conservative approximation cannot show that the velocity changes, even when the acceleration is large. We cannot prohibit these small transitions, because they might represent the last labeling W_k in the chain. If the labeling W has length $k + \epsilon$ for some integer k and $\epsilon < 1$, the length of W_{k+1} will be ϵ, which could be arbitrarily small.

One fairly straightforward way to deal with this exploits non-determinism. An additional "sink" state $\perp$ is introduced, so that $S(\widehat{m}) = \mathcal{Z} \cup \{\perp\}$. The transition relation is restricted so that it contains no transitions $(\widehat{W}, \widehat{s}, \widehat{s'})$ for which $\widehat{s} = \perp$. Thus, if $\perp$ appears in a chain, it must be the final state $\widehat{s}_k$. A transition (W, s, s') in the original machine for which W_x and $W_{\dot{x}}$ are shorter than one, are represented by a transition from $\rho(s)$ to $\perp$. We can now restrict transitions from integer states $\widehat{s}$ to states $\widehat{s'}$, to those satisfying $\dot{l} \le \widehat{s'} - \widehat{s} \le \dot{u}$. Transitions from any state $\widehat{s}$ to $\perp$ are allowed for any "input" provided that an integer $\widehat{s'}$ exists which satisfies the conditions developed above for the original approximation.

We have developed a discrete conservative approximation of the continuous integrator. The approximation is based on functions abstracting the input traces and the state space. It was constructed incrementally, starting with a complete state-transition relation. As each constraint was added, we confirmed that the abstraction functions mapped every chain of the original integrator to a chain of the approximation. This technique allowed us to construct simple arguments for correctness, focusing on one issue at a time. While the result is quite complex, it is the conjunction of conditions, each of which can be understood independently.

5.3 Composing Conservative Approximations

In Section 3, we defined the operations parallel composition and restriction for trace-automata. Using these operations, we combined a number of such automata to create a model of the dynamics of our train-set. We have just shown how to develop conservative approximations for the component automata. Now, we show that such approximations can be combined to yield an approximation of the entire model.

We begin this rather technical section with some lemmas that establish a relationship between the language accepted by a composite trace-automata, and the languages accepted by its components.

Lemma 5.3 *If w is in $L(m_1 \| m_2)$ then $w|_{A(m_1)}$ is in $L(m_1)$ and $w|_{A(m_2)}$ is in $L(m_2)$.*

Proof: Follows directly from the definitions of language acceptance and automata composition. □

Lemma 5.3 says that, when restricted in the obvious ways, any labeling W accepted by the composition $m_1 \| m_2$ is accepted by the individual components. One might hope that this implication could be strengthened to equivalence, but in general this is not the case.

To see this, imagine two stateless trace-automata m and n, each of which read $\mathcal{N}$-traces from an input tape a. Automata m consumes the symbol x from its tape, once symbol at a time:

$$(T, s, s') \in \Delta(m) \stackrel{\text{def}}{\equiv} T_a = \langle x \rangle$$

On the other hand automata n consumes the symbol x from its tape, but insists on receiving them two at a time:

$$(T, s, s') \in \Delta(m) \stackrel{\text{def}}{\equiv} T_a = \langle x, x \rangle$$

The labeling which associates the trace $\langle x, x, x, x \rangle$ with a is in the language of both automata, yet this incompatibility between the rates at which they consume their inputs prevents the composition $m \parallel n$ from accepting anything but the empty trace. Before proceeding further, we give a technical definition of compatibility:

Definition 5.4 *Compatibility: Let m and n be trace-automata and let $C = A(m) \cap A(n)$ be the variables that they have in common. We say that m and n are compatible if and only if for every pair of labelings, $U \in L(m)$ and $V \in L(n)$, such that $U|_C = V|_C$, there is a natural number k, and chains*

$$s_0 \xrightarrow{U_1} s_1 \xrightarrow{U_2} \cdots \xrightarrow{U_k} s_k$$

and

$$t_0 \xrightarrow{V_1} t_1 \xrightarrow{V_2} \cdots \xrightarrow{V_k} t_k$$

of length k belonging to m and n respectively such that each $U_i|_C = V_i|_C$.

Lemma 5.5 *let m_1 and m_2 be compatible trace-automata. Let w be an arbitrary labeling of $A(m_1 \parallel m_2)$. If $w|_{A(m_1)} \in L(m_1)$ and $w|_{A(m_2)} \in L(m_2)$, then $w \in L(m_1 \parallel m_2)$.*

Proof: Follows directly from the definition of compatibility. □

Theorem 5.6 *Composition: If, under the abstraction $(\Gamma\psi)$, $\widehat{m}$ is a conservative approximation of m, and $\widehat{n}$ is a conservative approximation of n, and if $\widehat{m}$ and $\widehat{n}$ are compatible, then $\widehat{m} \parallel \widehat{n}$ is a conservative approximation of $m \parallel n$ under the same abstraction $(\Gamma\psi)$.*

Proof: Let W be any labeling in $L(m \parallel n)$. We must show that $\psi(W) \in L(\widehat{m} \parallel \widehat{n})$. From Lemma 5.3 we can conclude that $W|_{A(m)} \in L(m)$ and $W|_{A(n)} \in L(n)$. Since $\widehat{m}$ is a conservative approximation of m under $(\Gamma\psi)$, the abstraction $\psi(W|_{A(m)})$ — hence, by proposition 5.1, $\psi(W)|_{A(m)}$ — must be in $L(\widehat{m})$. Similarly, we must have $\psi(W)|_{A(n)} \in L(\widehat{n})$. Since $\widehat{m}$ and $\widehat{n}$ are compatible, we conclude by lemma 5.5 that $\psi(W) \in L(\widehat{m} \parallel \widehat{n})$ as required. □

For example, the train model, shown in Figure 2 consists of the composition of several machines. Using techniques such as those developed in the last section, one can construct compatible conservative approximations of each machine independently. We can then form a conservative approximation of original composition by composing the approximations of the components.

The original machine involved several restrictions, so that only "interesting" labels were included in its behaviour. For example, the model is restricted so that the acceleration a of train is not visible. The following theorem allows us to similarly restrict the approximations.

Theorem 5.7 *Restriction: If $\widehat{m}$ is a conservative approximation of m under $(\Gamma\psi)$, and B is any subset of $A(m)$ then $\widehat{m}|_B$ is a conservative approximation of $m|_B$ under $(\Gamma\psi)$.*

Proof: Let V be any labeling of B in $L(m|_B)$. By definition, there must be a labeling W in $L(m)$ such that $W|_B = V$. Let W be such a labeling. Since $W \in L(m)$ and $\widehat{m}$ is a conservative approximation of m under $(\Gamma\psi)$ we must have $\psi(W) \in L(\widehat{m})$, hence $\psi(W)|_B \in L(\widehat{m}|_B)$. Since $V = W|_B$, we can conclude from proposition 5.1 that $\psi(V) = \psi(W)|_B$. Thus, $\psi(V) \in L(\widehat{m}|_B)$ as required. □

In this way, we can form a conservative approximation of the entire system. We begin by forming approximations of each component. These approximations components are then composed. Finally, the set of visible variables are restricted.

The resulting approximation captures the general behaviour of the system. It is, however, only an approximation; it may be too conservative to yield the verification results that we want. Suppose we use only independent approximations of the components of our train model. The resulting model is too conservative to conclude that a train will stop if the brakes are applied. The best that one can conclude is that the train will slow down to some minimum speed. The problem is that the state 0 in the integrator representing velocity, abstracts a range of states in the original model. Doubtless, there are numerous ways that the abstraction functions could be altered to address the problem with this specific model. We offer a more generally applicable solution.

The essence of the problem is is that there is a distinct topological feature in the phase-space of the train dynamics. While the control signal is `STOP`, there is a fixed point attractor at $v = 0$ with a basin of attraction, $v > 0$. As long as the velocity is positive, and the control signal remains `STOP` the train will eventually stop. Correctness depends on the precise location of this fixed-point. For example, if the attractor were at $v = \frac{1}{2}$, the system would not be correct, since trains could not be stopped.

Returning to the original hybrid model, we examine its behaviour close to this fixed-point. If the velocity of a train is less than k_{stop} at time t, and if the control signal is `STOP` from time t to time $t+1$, then the velocity will be 0 from time $t+1$ until the control signal is `GO`. It is straightforward to build a conservative approximation that captures only this behaviour. The machine will have input variables for the velocity, (l, u), and the control signal ctl. The machine will have two states `SLOW` and `FAST` which summarize events in the previous time-period. If the velocity in the previous time-period did not exceed k_{stop} and the control signal was `STOP`, then the machine will begin the next time period in state `SLOW`. Otherwise, the machine could be in either state. If the machine begins a transition in state `SLOW`, and the control signal is `STOP`, then the velocity upper bound must be 0. Otherwise, the velocity is unconstrained.

The machine is a conservative approximation of the system. If the system is not close to the fixed-point ($u >= k_{\text{stop}}$), then its behaviour is not constrained by the model. On the other hand, if the system is close to the fixed point for a full time unit, ($u <= k_{\text{stop}}$) then the approximation ensures that the fixed point is reached at the end of the interval.

The following corollary to Theorem 5.6 allows us to compose this conservative approximation of the fixed point, with the more general one obtained from the components.

Corollary 5.8 *If m_1 and m_2 are compatible conservative approximations of m, under the same abstraction function, then so is their composition.*

Proof: The corollary follows from the observation that $m \,||\, m$ is identical to m. □

This technique is similar to that which we used to construct the approximation of the integrator. In that case, we constructed the final approximation by combining independent constraints on the state transition relation. Here, we are composing conservative approximations that were derived independently. The machine that we are approximating, the velocity control system for a train, is itself a composition. One of the approximations is derived by composing approximations of the components. The other was derived by hand. This was possible to do because it needs to capture only a very small, simple part of the overall behaviour of the system.

5.4 Finite Approximations

So far, our approximations have been discrete, but not finite. The integrator approximation, for example, has neither a finite state-space, nor a finite input alphabet. However, it is simple to construct a finite version. We adjust the state-mapping ρ and the abstraction function ψ so that they map onto finite subsets of, respectively, the integers and the integer-bounded intervals. For example, we could choose integer values min and max, and adjust ρ to map every state with value max or greater onto the state max in the approximation, and every state with value min or less onto min. Similarly, we map subtraces whose value exceeds max onto intervals with upper bounds $u = max$ and traces with values less than min to intervals with min as their lower bound. The result will be a model with a finite next-state relation. Of course the model will be unable to distinguish between values greater than max. Nor will it be able to distinguish between values less than min. As a result, it will not be useful for verifying properties which depend upon such distinctions. In our train model, the values of all traces are bounded, so we merely have to choose min and max to be outside these bounds.

6 Verification Using Approximations

In the previous sections, we showed how to construct a discrete conservative approximation of our system. The resulting approximation is a non-deterministic finite-state machine, that reads one symbol from each of its many input tapes during each state-transition.

To demonstrate the feasibility of this approach, we have verified the system described in this paper. The system is modeled using the Voss [11] verification system. Voss provides a functional language with built in Ordered Boolean Decision Diagrams[2] (OBDDs). The language allows the redefinition of standard infix operators. We have used this facility to re-define the standard arithmetic and comparison operators using two's complement bit-vector arithmetic. The next-state-relations for the various component approximations can be expressed straightforwardly in this environment. Such relations are essentially sets of labeled transitions which can be represented using familiar techniques[3]. The source and destination state-spaces are represented by the set of possible assignments to disjoint sets of boolean variables. A third set of variables represents the labels. A set of transition is expressed as a boolean valued function over the union of these three sets.

The Voss system represents such an expression internally using OBDDs. Composing two machines amounts to computing the boolean conjunction of the next-state relations for their components. Domain restriction amounts to existentially quantifying the variables representing the hidden input tapes.

Using this technique, we have built an OBDD representation of the complete system, with a track length of 256, a maximum velocity of 6, an acceleration, k_{GO} of $\frac{1}{2}$, and

k_{stop} of 3. The controller measures positions with an precision of 8 track units — that is it can determine in which of eight equal length sections a train is located. Using a suitable variable ordering, we were able to represent the transition relation for the entire system using less than twelve-thousand OBDD nodes. The Voss code which generates this representation is included in Appendix B.

We wished to verify that when started at opposite positions on the track, the two trains would never collide. This amounts to saying that the hybrid model accepts no traces during which the positions of the two trains are simultaneously equal. We translate this specification into our discrete framework, using the abstraction function $(\Gamma\sigma)$. Thus, in the discrete framework we must verify that the model accepts no traces for which the ranges $(v.l, v.u)$ of tokens representing the positions of the two trains intersect. Once the specification has been established, verification is straightforward using standard model checking techniques.

Both the construction of the model and the verification were done in the most simple-minded way. For example, the definition of the integrator approximation used in the model is a clause by clause translation of the constraints developed in Section 5.2. The entire transition relation is constructed by conjoining approximations like this, and then existentially quantifying out the variables that represent hidden inputs. The verification was done by chaining forward from the start-state, computing the reachable state-set, and then checking to ensure that a hazardous transition could not be taken from it. Even so, with suitable variable orderings, the entire process takes less than 15 minutes on a Sparc-10 server.

7 Conclusions and Future Work

In this paper, we have presented an approach to the the verification of hybrid systems. We model the system initially using *trace-automata*, a framework in which continuous, discrete, and hybrid systems can be expressed. Subsequently, a discrete *conservative approximation* of this model is build in this same framework. The resulting discrete trace automata is verified against an abstraction of the original hybrid specification using conventional techniques.

The trace automata formalism has been designed to facilitate our investigation into techniques for modeling and verifying hybrid systems. In this investigative spirit, the trace-automata framework is a general as possible. In particular, the formalism allows multiple time-scales, with no *a priori* relationship between them. Of course such a relationship can be established in the context of a particular model. This freedom makes it relatively easy to construct models with no obvious physical interpretation. A practical modeling system based on this theory, might well introduce additional restrictions on the way in which models can be constructed and combined.

As an illustration we have presented a model of a simplified version of the train in our lab. We then verified that the control mechanism will not allow two trains to collide. The example system, a simplified version of our model train set, consists of several parts, each with different behaviours. The trains themselves behave continuously according to the laws of physics. The controller is essentially discrete. Each part is modeled independently by a *trace automata*. The models of the parts are then composed, to form a model of the complete system.

We introduced a simple specification framework based on sets of prohibited behaviours which we call failure sets. We then developed an abstraction function, which mapped from failure sets in the continuous domain, to failure sets in a discrete domain. Once this abstraction function was defined, we showed how to construct discrete approximations of the components in our train model, so that they were conservative with respect to it. This process was illustrated in some detail, by developing a conservative approximation of an integrator.

An approximation of the entire system was constructed by performing the same composition operations on these primitive approximations that were performed on the originals. The operations that were used preserve the conservative approximation relationship for our specification language. Thus, we were able to conclude that the resulting discrete finite-state automata was a conservative approximation of the original continuous automata.

In fact, the resulting model was too conservative to obtain the desired verification result. The control algorithm exploits the precise location of a topological feature in the phase space. Specifically, trains have a fixed-point attractor at $v = 0$. Put another way, they eventually stop when the brakes are applied. Our approximation could not differentiate between the condition of being stopped, and moving very slowly. Once this problem was identified, however, it was trivial to construct a conservative approximation that captured this behaviour. In doing so, we needed only to concern ourselves with the local behaviour of the train close to this fixed point. Other behaviours were captured precisely enough for our verification task by the existing model.

Finally, we were able to use this discrete model to verify that the original model was collision free. The specification was translated from the original continuous domain to the discrete domain of the approximation by the abstraction function developed earlier. Conventional model-checking techniques were used to show that the discrete model satisfied this abstract specification. Since the discrete model was know to be a conservative approximation of the original with respect to this abstraction function, we concluded the original model would not admit collisions.

In principle, the process of composing and connecting the primitive approximations could have been accomplished automatically. We envision a tool, with a built-in set of primitive continuous models and their (parameterized) approximations. The user would construct a model of the system, by composing and connecting these built-in primitives. The tool would automatically perform the same compositions and connections on the primitive approximations, producing a conservative approximation of the entire system, suitable for verification. Identifying a useful set of primitives, and appropriate approximations of them remains a matter for future research.

For the purpose of illustration, we have limited the expressiveness of our specification framework. These limitations, however, are more historical than fundamental to the approach. Indeed developing approximations for other types of specifications seems like a natural way to extend the work. Here it seems likely that one will wish to develop different approximations for different kinds of questions. For example it may well be best to develop one approximation that is conservative with respect to safety questions, and another that is conservative with respect to liveness properties.

We are suggesting an approach to hybrid system verification, which essentially reduces the problem to that of verifying discrete systems. The latter, has been extensively studied, and considerable progress has recently been made. One possible objection to this approach is its failure to exploit the underlying continuity of the physical systems being modeled. As it stands, this continuity is exploited, if at all, only in the development of primitive approximations. Whether the continuity of an underlying system can be exploited to simplify the verification of its discrete approximation, remains an intriguing question for future research.

A A summary of notation

Symbol	Meaning	Page
$\|w\|$	The length of trace w.	5
(Π, Σ, Φ)	Defines a trace-automata domain. Π is the set of variables in the domain, Σ associates each name $a \in \Pi$ with a domain of values Σ_a. Φ associates each name $a \in \Pi$ with a time domain, Φ_a which must be either the reals or the naturals.	6
A^D	The set of labelings of A in the trace-automata domain D	6
$W\|_B$	W restricted to B — W is a labeling of some set of variables A, of which B is a subset. $W\|_B$ labels each variable in B with the same trace as W does.	6
$A(m)$	The variable-set of the trace automaton m.	6
$S(m)$	The state-set of the trace automaton m.	6
$I(m)$	The initial states of the trace automaton m: $I \subseteq S$.	6
$\Delta(m)$	The transition relation of trace the automaton m: $\Delta(m) \subseteq A(m)^D \times S(m) \times S(m)$	6
$L(m)$	The language accepted by m	7
$m_1 \\| m_2$	The composition of m_1 and m_2.	10
$m\|_B$	The trace automaton m restricted to B. The restriction $m\|_B$ accepts a trace W over B if it is the restriction of a trace accepted by m.	10
$\mathcal{M}$	The set of models in a verification domain.	13
$\mathcal{Q}$	The set of specifications in the verification domain.	13
$\models$	The satisfaction relation in a verification domain — $\models \subseteq \mathcal{M} \times \mathcal{Q}$.	13
Ψ	A generic abstraction function from specifications to specifications. Typically Ψ would be generated from a function ψ by applying the Γ operator.	13
$G_\Psi(m, \widehat{m})$	The set of specifications on which m and $\widehat{m}$ agree.	13
$\widehat{m}_1 \preceq_\Psi \widehat{m}_2$	The automaton $\widehat{m}_2$ is as good an approximation as m_1 under the abstraction function Ψ.	14
F	Failure set — A set of prohibited labelings.	15
ψ	A generic separable abstraction function from labelings to labelings.	15
Γ	The operator that generates a function mapping sets of labelings to sets of labelings, from a function ψ mapping labelings to labelings.	15
σ	The specific separable abstraction function developed for the train set example.	16
ρ	The state abstraction function that was used to build the integrator approximation.	17

B A Discrete Conservative Approximation of the Train System

```
//conservative approximation of an integrator
let integrator (s,s') (ldot,udot) (l,u) =
    (ldot >= '+0 IMPLIES
        u ISBETWEEN (s',s'++) AND
        l ISBETWEEN (s--,s)) AND
    (udot <= '+0 IMPLIES
        u ISBETWEEN (s,s++) AND
        l ISBETWEEN (s'--,s')) AND
    (udot > '+0 AND ldot < '+0 IMPLIES
        u ISBETWEEN (s,s+udot++) AND
        u ISBETWEEN (s',(s'-ldot)++) AND
        l ISBETWEEN ((s+ldot)--,s) AND
        l ISBETWEEN ((s'-udot)--,s')) AND
    (sink ~ NOT_SINK IMPLIES
        s' - s ISBETWEEN (ldot,udot)) AND
    (sink ~ SINK IMPLIES
        (s' >= s AND s' - s <= udot) OR
        (s' <= s AND ldot <= s' - s));

// A "integrator" for which state and output are "mod m" integers.
let modintegrator  L (s,s') (ldot,udot) (l,u) =
    (ldot >= '+0) AND (udot <= L) IMPLIES
        (s' > s IMPLIES s' - s <= udot) AND
        (s' < s IMPLIES s' + L - s <= udot) AND
        (s' ~ s IMPLIES udot ~ '+0) AND
        u ISBETWEEN (s', s'++ % L) AND
        l ISBETWEEN (s-- % L, s);

// conservative approximation of the "GO" differential equation
let difeqnGO (yl,yu) (xl,xu) =
    // a = 1/2 (MaxV - v)
    let ceilF x = half ((MaxV - x) ++) in
    let flrF x = half (MaxV - x) in

    (xl ~ xu IMPLIES yl ~ flrF xl AND yu ~ ceilF xu) AND
    (xl < xu IMPLIES
        yl ISBETWEEN (flrF xu, flrF (xu--)) AND
        yu ISBETWEEN (ceilF (xl++), ceilF xl));

// conservative approximation of the "STOP" differential equation
let difeqnSTOP (yl,yu) (xl,xu) =
    // v=0 ? a = 0 | a = -k2 MaxV
    let a = Kstop in
    (yu ~ (choose (xl <= '+0) ('+0) a)) AND
    (yl ~ (choose (xu <= '+0) ('+0) a));
```

```
// conservative approximation of the "switch"
let mSwitch ctl (a1l,a1u) (a2l,a2u) (al,au) =
    (ctl ~ STOP IMPLIES a1l ~ al AND a1u ~ au) AND
    (ctl ~ GO IMPLIES a2l ~ al AND a2u ~ au);

// conservative approximation of the behaviour near the "fixed point"
let mFixedPoint (fps, fps') ctl (vl,vu) =
    (fps' ~ '+1 IMPLIES vu >= (negate Kstop) OR ctl ~ GO) AND
    (ctl ~ STOP AND fps ~ '+0 IMPLIES vu ~ '+0);

// Composition of the equations with approximations of
// the switch, integrator, and fixed-point
let mVel vs fps ctl (vl,vu) =
    exists [al,au] (
        exists [a1l,a1u,a2l,a2u]  (
            difeqnGO (a2l,a2u) (vl,vu) AND
            difeqnSTOP (a1l,a1u) (vl,vu) AND
            mSwitch ctl (a1l,a1u) (a2l,a2u) (al,au)) AND
        integrator vs (al,au) (vl,vu) AND
        mFixedPoint fps ctl (vl,vu));

// composition of the position integrator with mVel
let mPos  vs fps ps ctl (vl,vu) (pl,pu) =
    exists [vl,vu] (
        mVel vs fps ctl (vl,vu) AND
        modintegrator L ps (vl,vu) (pl,pu));

// approximation of the control algorithm for train 1
let m1ctl  =
    let maxp = (L++) / cdiv in
    let nsStop = (((c1i ++) % maxp) ~ c2i) OR (c1i ~ c2i) in
    (c1' ~ (choose nsStop STOP GO)) AND
    (c1o ~ c1);

// approximation of the control algorithm for train 2
let m2ctl  =
    let maxp = (L++) / cdiv in
    let nsStop = (((c2i ++) % maxp) ~ c1i) OR (c2i ~ c1i) in
    (c2' ~ (choose nsStop STOP GO)) AND
    (c2o ~ c2);


// approximation of the position sampling process
let sample x (xl,xu) =
   let xl = xl / cdiv in
   let xu = xu / cdiv in
   (xl <= xu IMPLIES x ISBETWEEN (xl,xu)) AND
```

```
   (xl > xu IMPLIES  (xl <= x) OR (x <= xu));

// approximation of the controller (m1ctl,m2ctl,sample(1) and sample(2))
let cmodel =
    exists [c1i,c2i]
        (m1ctl AND m2ctl AND
            (sample c1i (p1l,p1u)) AND
            (sample c2i (p2l,p2u)));

let next_state_relation =
    let m1 = exists [c1o]
        (cmodel AND
            mPos (v1,v1') (fp1,fp1') (p1,p1') c1o (v1l,v1u) (p1l,p1u)) in
    exists [c2o]
        (m1 AND mPos (v2,v2') (fp2,fp2') (p2,p2') c2o (v2l,v2u) (p2l,p2u));

let initial_state =
    (v1 ~ '+0) AND (v2 ~ '+0) AND (p1 ~ '+0) AND (p2 ~ '+128) AND
    (c1 ~ GO)  AND (c2 ~ GO);

let begin_state_vars  = [v1 , fp1 ,p1 ,c1 ,v2 ,fp2 ,p2 ,c2];
let end_state_vars = [v1', fp1',p1',c1',v2',fp2',p2',c2'];
let input_vars  = [p2u,p2l,p1u,p1l];

// returns c AND ((xl,xu) intersects (yl,yu))
let intersects (xl,xu) (yl,yu) =
    (xl <= xu AND yl <= yu AND yl <= xu AND xl <= yu) OR
    (xl > xu AND (yu > xl OR yl < xu)) OR
    (yl > yu AND (xu > yl OR xl < yu));

// The hazard_set is a set of states from which an "unsafe" transition
// can occur.
let hazard_set =
  let nsr = exists end_state_vars next_state_relation in
  exists input_vars (nsr AND (intersects (p1l,p1u) (p2l,p2u)));
```

References

[1] R. Alur, C. Courcoubetis, T. A. Henzinger, and P. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In R. Grossman, A. Nerode, R. Ravn, and H. Rischel, editors, *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, pages 209–229. Springer-Verlag, 1993.

[2] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[3] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.

[4] Jerry R. Burch. *Trace Algebra for Automatic Verification of Real-Time Concurrent Systems*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA 15213, August 1992.

[5] Edmund M. Clarke, Orna Grumberg, and David E. Long. Model checking and abstraction. In *Procedings 19th Annual ACM Symposium on Principles of Programming Languages*, January 1992.

[6] R. P. Kurshan and K. L. McMillan. Analysis of digital circuits throught symbolic reduction. *IEEE Transtactions on Computer-Aided Design*, 10(11):1356–1371, November 1991.

[7] Anthony McIsaac. A formalization of abstraction in lambda. In Carl Seger and Jeffrey Joyce, editor, *HUG '93 HOL User's Group Workshop*, pages 229–240. University of British Columbia, Department of Computer Science, August 1993.

[8] G. H. Mealy. A method for synthesizing sequential circuits. *Bell System Technical Journal*, 34(5), May 1955.

[9] E. F. Moore. Gedanken experiments on sequential machines. In *Automata Studies*. Princeton University Press, Princeton, N.J., 1956.

[10] Anders P. Ravn, Hans Rischel, and Kirsten Mark Hansen. Specifying and verifying requirements of real-time systems. *IEEE Transatcions on Software Engineering*, 19(1):41–55, January 1993.

[11] Carl-Johan H. Seger. Voss — a formal hardware verification system user's guide. Technical Report 93-45, Department of Computer Science, The University of British Columbia, 1993.

[12] Ying Zhang and Alan K. Mackworth. Constraint nets: A semantic model for real-time embedded systems. Technical Report 92-10, University of British Columbia, Vancouver, B.C., Canada, October 1992.

[13] Ying Zhang and Alan K. Mackworth. Will the robot do the right thing? Technical Report 92-31, Department of Computer Science, The University of British Columbia, November 1992.