Electronic Notes in Theoretical Computer Science 50 No. 1 (2001) – Proc. ATMOS 2001
URL: http://www.elsevier.nl/locate/entcs/volume50.html

Designing a Controller for a Multi-Train Multi-Track System [1]

Deepak Kapur [2]

University of New Mexico

Victor L. Winter and Raymond S. Berg [3]

Sandia National Laboratories

Abstract

The use of formal methods for analyzing and synthesizing a controller for a multi-train multi-track railway system is discussed. The research was motivated by a case study involving the *Bay Area Rapid Transit* (*BART*) system. The overall goal is to design a train acceleration control function that enables trains to be safely placed but also increases system throughput. The use of a modeling language for specifying safety properties and a control function is illustrated. The program transformation methodology supported in the *HATS* system is employed to generate an efficient implementation from a high-level specification of a controller. This implementation can then be used to simulate the controller behavior, thus further enhancing confidence in the design. Properties of optimization transformations can be verified using an rewrite-rule based induction theorem prover *Rewrite Rule Laboratory* (*RRL*).

1 Introduction and Motivation

An overview of the use of formal methods and program transformation methodology in analyzing and synthesizing a controller for a multi-train multi-track system is presented. This research was motivated by a case study involving the Bay Area Rapid Transit (BART) system [10]. The overall objective of the BART case study, briefly discussed in the next section, is to develop train (acceleration) control functions, enabling trains to be safely spaced closer together, thereby increasing the systems throughput. Our main objective has been the development of a suitable model, allowing the formal definition of safety properties required by such a system, and to then reason about the behavior of train models with respect to these properties.

[1] This work was supported by the United States Department of Energy under Contract DE-AC04-94AL85000. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy.

[2] Deepak Kapur was also partially supported by NSF grant nos. CCR-9996150 and CDA-9503064. Email: kapur@cs.unm.edu

[3] Email: vlwinte@sandia.gov, rberg@sandia.gov

The paper is organized in the following way. After giving an overview of the BART system, the formal model and the modeling language are discussed; more details can be found in [5]. Two key constructs, *profiles* and *constraints* are introduced. A profile is a sequence of tuples, where each tuple consists of position element and an associated speed element. A constraint builds on the concept of a profile, by using the elements in the profile to define a satisfiability region in the position-speed plane. Trains, tracks, stations and signals can be all modeled using profiles, and their related safety properties can be modeled by corresponding constraints. A train profile is indexed by a global time clock. The behavior of a train is governed by the behavior of the train ahead of it on the track. Profiles are discussed in discrete as well as continuous terms with respect to position and speed. Discrete profiles are necessary because that is how data is obtained from sensors. Continuous profiles are introduced for two reasons: (i) the physical train system is continuous, and (ii) thus, most safety properties can be directly stated in terms of constraints based on continuous profiles. This is followed by a discussion of mappings between discrete and continuous profiles. Assumptions made for defining these mappings and their justification vis a vis safety properties are discussed. Section 4 discusses the design of a controller algorithm. A specification of the control function is given. This specification is then transformed to an efficient implementation using *HATS*, a program transformation system being developed by the *High Integrity Software* (*HIS*) program at Sandia National Laboratories. A brief overview of HATS is given in Section 5. Some of the features of the system are illustrated by discussing an optimizing transformation used on the specification of the controller. Section 6 is on the validation of the model using a combination of automated reasoning tools and simulation. A rewrite-rule based induction theorem prover *Rewrite Rule Laboratory* (*RRL*) [4] was used for verifying the correctness of optimizing transformations used to generate an efficient implementation from a specification; Simulation was used to enhance confidence in the design by executing different scenarios.

It should be mentioned that most of the discussion in this extended abstract is informal because of space limitations; an interested reader can get more details including the formal definitions, precise statements of theorems and properties as well as proofs by consulting [13].

2 Overview of BART

BART provides heavy commuter rail service in the San Francisco Bay Area. Typically over 50 trains, most consisting of 10 cars, are in service. Cars are driven by electric motors powered by a 1000 VDC, “third rail”, and use both regenerative and friction brakes. The system is controlled automatically, and on-board operators have a limited role in normal operations (e.g., they signal the system when the platforms are clear so a train can depart a station, and perhaps, in some exceptional circumstanaces, they pull the emergency brake).

With a few minor exceptions, the BART system consists of double track:

one track going one direction and one track going the other (i.e., the track is not a loop.) An acceleration controller is at the front and back of each train. At the end of the line, the front and back controllers are redefined, and the train goes in the other direction.

To serve more passengers, BART needs to utilize certain sections of the track more efficiently. Adding new tracks to these sections, e.g., a tube under the bay and underground in the heart of San Francisco, would be prohibitively expensive. Because of these considerations, a decision has been made to address the throughput problem by spacing the trains closer together.

2.1 Interlocking

Another major aspect of train control is interlocking - the management of track switches and associated signals for entering track segments. At BART, interlocking is handled by a system separate from the train (acceleration) control. The train controller will simply see "go" or "stop" indicators at various track locations, and should enter a "gate" only if allowed. It is the responsibility of the interlocking system not to move a switch if a signal change occurs when it is too late for an approaching train to stop. Stopping at a station platform is much like stopping in front of a gate, although there are some additional controls to assure the final stop is at the precise location.

3 Formal Model

We define a multi-track, multi-train system as a system consisting of one or more trains sharing non-overlapping tracks. Trains must run on tracks subject to certain constraints, such as stopping at designated stations and signals when appropriate. We develop a model of a single train running on a single track. Using compositionality, the model easily extends to multiple trains on a single track as well as to multiple tracks. Therefore, it suffices to consider a given train (called the *object* train) henceforth, and analyze its behavior relative to the train ahead of it on the track (called the *lead* train).

Though this model is an approximation of an actual train system, it nevertheless captures the salient features of the train such as how quickly the train can change from one acceleration to another as well as acceleration limits. These values can be determined from the current speed and acceleration of the train. By capturing this kind of information, the model reflects the physical limits that are encountered with respect to the movement of large heavy objects driven by electric motors, such as trains.

The sense/react cycle time also reflects these limits due to the fact that using a particular train technology implies a certain granularity in the control of the train. Similarly, the maximum speeds for track segments together with a set of safety constraints directly influence how long a train can go unsupervised and hence also constrain the duration of the sense/react cycle. For our analysis, discrete global clock time is measured in this unit; for simulation, it is assumed to be a $\frac{1}{2}$ second interval.

The following events are defined to be disastrous.

(i) A train collides with another train.

(ii) A train fails to halt for a signal or station.

(iii) A train exceeds the track speed limit (risks derailment).

(iv) A train unnecessarily performs an emergency stop. Note however, that we will consider it to be acceptable for a train to perform an emergency stop in response to abnormal environmental conditions (e.g., derailment of the train immediately in front of the train).

It is assumed that sensors, describing the state of a train, are correct within certain known margins of error, acceleration commands are correctly realized by the motor, etc.; under these assumptions, events (i), (iii), and (iv) can be avoided by a correct control function. Event (ii) can be avoided assuming signals are set by the interlocking system at appropriate times.

The following assumptions are made about train behavior:

- A train cannot stop instantaneously; instead, the train takes time and distance to stop based on its speed and acceleration. Derailment of the lead train can however take place instantaneously in time and position (see discussion below about conservative assumptions of the model).
- Acceleration values must be "ramped up/down" over time.
- Limited assumptions can be made regarding the behavior of a lead train (e.g., it may derail).

Given the train speed, s, and current acceleration, acc, a function, $\mathcal{N}\_\mathcal{A}(s, acc)$, listing possible acceleration values in the next time cycle based on the physical model of the train is assumed to exist (see [10] for a detailed explanation of such a function).

3.1 Profiles

Two constructs, *profiles* and *constraints*, model the components in the train system. A profile describes a relation between position and speed, where position and speed are indexed by discrete (continuous) time for a discrete (continuous) profile. We assume that there is a global clock, shared by all trains, signals and stations on a given track. Given a time t and train i, the profile of the i^{th} train gives its position and speed at time t. Profiles can be used to model discrete/continuous trajectories in the position-speed plane.

Components in the train system are modeled as follows:

- *train state* - A train state consists of a position, speed, acceleration triple.
- *train behavior* - A sequence of train states whose position is monotonically increasing. States in a behavior sequence are indexed by discrete time, and thus describe the state of a train at a specific point in time.
- *discrete train profile* - A discrete train profile is a sequence of position-speed tuples, and is obtained from a train behaivor by dropping the acceleration argument from the behavior's train states. Elements of a discrete train profile are indexed by discrete time.

- *continuous train profile* - A continuous train profile is a set of position-speed-time triples, in which position, speed, and time are continuous over a bounded range of values.[4]
- *track profile* - The track is modeled as a sequence of position-speed tuples that is monotonically increasing on position, specifying the position on the track where the corresponding speed limit goes into effect. This speed limit extends up to the position of the next track segment tuple. We assume track segments have constant speed limits associated with them and therefore are invariant over time. This time-invariance allows us to compare discrete train profiles with track profiles in a meaningful way.
- *traffic-signals profile* and *station-signals profile* - These are modeled by sequences of position-speed tuples in which the speed associated with a position can fluctuate over time. For example, if a signal is green, the speed at the position of the signal is the maximum speed allowed on that track segment; if the signal is red, the speed at the position of the signal is 0. There are similar constraints for station signals. Within the model, it is assumed that the train interlocking system will operate appropriately, avoiding changing a signal in case a train is so close to the signal that it may not be able to obey the changed signal without using an emergency stop.
- *continuous component profile* - A continuous component profile is a set of position speed tuples that results when either a track profile, traffic-signals profile, or station-signals profile is used as the basis for constructing a continuous profile. In these types of continuous profiles, position and speed values are continuous and bounded over a range of values.

3.2 Safety: A Continuous Property

Because sensor outputs are measured at discrete time intervals, the components of the train system can be most directly modeled using discrete profiles. We are now faced with the task of formally defining various safety properties with respect to this model. The problem here is that safety properties are continuous properties that describe the behavior of the physical train system, and not the discrete view induced by the sense/react loop.

Safety properties could be defined indirectly by stating relationships between various discrete profiles. One must then infer from a given set of discrete relationships that the continuous property holds. For example, consider one of the easiest safety properties, *SP, a train should not exceed the speed limit of the track segment on which it is traveling.* A first attempt at stating this property in terms of our discrete model might result in the following:

Let $track = <(seg\_pos_1, seg\_speed_1), \cdots, (seg\_pos_n, seg\_speed_n)>$ denote a (discrete) track profile, and $train_i = <(p_1, s_1), \cdots, (p_m, s_m)>$ denote the profile of an object train i.

[4] We could have kept a continuous train profile as a position-speed tuple indexed by continuous time, much like a discrete profile.

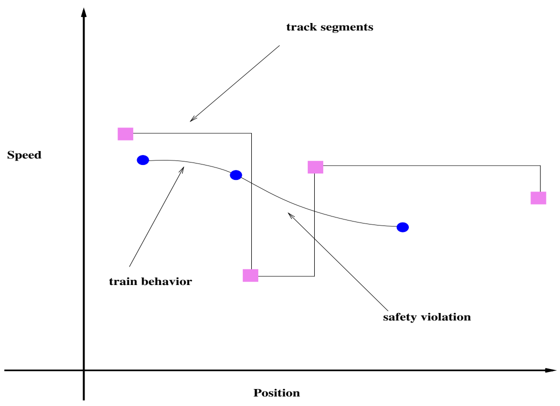

Fig. 1. Violation of Track Speed Limit

$safe\_speed(train, track) \stackrel{def}{=} \forall(1 \leq j < n \wedge 1 \leq t \leq m):$
$track[j] = (seg\_pos_1, seg\_speed_1) \wedge track[j+1] = (seg\_pos_2, seg\_speed_2) \wedge$
$train_i[t] = (p, s) \wedge (seg\_pos_1 \leq p < seg\_pos_2) \rightarrow s \leq seg\_speed_1.$

The question here is whether $safe\_speed(train, track) \Leftrightarrow SP$? As the continuous diagram shows, the predicate does not cover all the cases.

The *safe_speed* predicate given above can be extended to cover the case shown, but the question still remains: "Have all the cases been covered"? In a continuous framework, safety properties such as this can be directly stated. Assuming that the initial state of the object train satisfies the track segment speed limit, its behavior profile satisfies the constraints imposed by the track segment speed limit iff there does not exist a point in the position-speed plane belonging to both the continuous representation of the track and train behavior profile (i.e., an intersection does not occur).

Given the elegance with which safety properties can be stated in a continuous framework, we extend our model into this framework by defining a mapping to translate discrete profiles into continuous profiles and vice versa. This mapping makes explicit, a conservative understanding of what occurs in the gaps that exist between adjacent discrete points in our model (see Section 3.3), and it is with respect to this understanding that we formally define safety through the *satisfies constraint* operator $\ll$ as discussed in Section 3.5.

3.3 Continuous Train Profiles

The sensor information provided by the system gives us a precise (within a given margin of error) description of the state of a train at discrete $\frac{1}{2}$ second intervals. How should one model behavior between these discrete points? The problem is that the continuous behavior of trains between these discrete points falls outside of the resolution of our model. We address this model limitation by stating properties that continuous train profiles must satisfy in order to be consistent with the discrete sensor readings provided by the system, thus, in effect bounding such behaviors.

Let $tp =< (p_1, s_1), \cdots, (p_n, s_n) >$ denote a discrete profile corresponding to

a train behavior from (discrete) time $t = 1, \cdots, n$. Let *ctp* denote a continuous train profile $\{(p, s, t) \mid t \in [1, \cdots n]\}$; note that the third component in the triple is time ranging over a continuous time line between 1 and n.
$consistent(ctp, tp) \Leftrightarrow \forall (p, s, t_1) \in ctp, t_2 \in \{1, 2, ..., n\} : (t_1 = t_2 \wedge tp[t_2] = (p_1, s_1) \rightarrow p = p_1 \wedge s = s_1) \wedge (t_2 < n \wedge t_2 < t_1 < t_2 + 1 \wedge tp[t_2] = (p_1, s_1) \wedge tp[t_2 + 1] = (p_2, s_2) \rightarrow (p_1 < p < p_2 \wedge min(s_1, s_2) \leq s \leq max(s_1, s_2))$.

Definition 3.1 $\mathcal{CP}_{tp} = \{ctp \mid consistent(ctp, tp)\}$.

For a given discrete train profile, *tp*, $\mathcal{CP}_{tp}$ bounds the type of continuous profiles which an actual train may have while still producing sensor readings that are consistent with *tp*.

3.4 Continuous Component Profiles

Let pf denote a (discrete) track, signal, or station profile whose ordering of tuples is strictly monotonic on position with respect to the $\leq$ relation. The continuous component profile, *ccp*, corresponding to pf is defined as:

$$\begin{aligned} ccp_{pf} = \{(p, s) \mid \exists k : pf[k] = (p_1, s_1) \wedge pf[k+1] = (p_2, s_2) \wedge \\ [(p_1 \leq p < p_2 \wedge s = s_1) \vee (p = p_2 \wedge (s_1 \leq s_2 \rightarrow s_1 \leq s \leq s_2) \wedge \\ (s_2 < s_1 \rightarrow s_2 \leq s \leq s_1)]\}. \end{aligned}$$

Definition 3.2 $\mathcal{CP}_{pf} = \{ccp_{pf}\}$.

3.5 Constraints on Profiles

A *constraint* $\mathcal{CP}_<$ defines a *satisfiable region* with respect to a given set $\mathcal{CP}$ of continuous profiles. A *satisfies constraint* operator $\ll$ defines the conditions under which a continuous profile relates to a constraint. Here we show the more general case where the set $\mathcal{CP}$ of continuous profiles is derived from a discrete train profile. The track and signals cases are essentially the same, except that the time component is not present in the tuples, and is ignored while making comparisons; the relation $\ll$ below is overloaded in this sense.

Definition 3.3 $\mathcal{CP}_< \stackrel{def}{=} \{(p, s, t) \mid \forall ctp \in \mathcal{CP}, (p, s_1, t) \in ctp : 0 \leq s < s_1\}$.

Definition 3.4 $ctp \ll \mathcal{CP}_< \Leftrightarrow \forall (p, s, t) \in ctp : (p, s, t) \in \mathcal{CP}_< \vee$
$\neg \exists s' : (p, s', t) \in \mathcal{C}_<$.

Given discrete profiles for an object train OT and its lead train LT, the *continuous stopping profile* of OT is computed by its controller under different scenarios: (i) LT stops normally, (ii) LT makes an emergency stop, and (iii) LT derails. Each scenario maps to a formal constraint which must be satisfied by any acceleration proposed by the OT controller.

Another constraint is induced by the track segment speed limit, which defines the maximum speed a train can safely travel on that segment. This safety requirement is stated as: $(ctp \ll \mathcal{CP}_{trp})$, where *ctp* is any continuous profile in $\mathcal{CP}_{tp}$, the set of continuous profiles consistent with a discrete train profile tp of OT, and $\mathcal{CP}_{track}$ is the set of continuous profiles associated with a discrete track profile trp. Similarly, safety properties originating from other components of the system can be formally stated.

Safety constraints can thus be specified in a compositional manner in terms of the safety property of trains running on each track because of the assumption that tracks do not interact/interfere with each other. The safety property of trains on a single track can be expressed in terms of the safety train of each object train running on the track. This property is specified, again in a compositional manner, in terms of the behavioral constraints imposed by the associated lead train, track, signals and stations.

As discussed in [5], the domain language consists of a hierarchy of constructs and definitions tied together by theorems characterizing their properties. This hierarchy is based on primitive constructs and operations dealing with enumerable sets and sequences. Giving these primitive constructs formal operational definitions, implies that the specifications written in this domain language are executable. However, generally speaking, the computation sequences defined by such specifications will typically be very inefficient.

The main reason for designing a special-purpose modeling language was to develop a high-level abstract specification which can be transformed using the transformations supported by the HATS system which we have been developing over a number of years. This approach has the distinct advantage of quickly generating a running implementation for simulation purposes to validate assumptions and debug the model. We are unaware of any modeling language for reactive systems (e.g. [8]) which can be used to quickly generate a prototype implementation from a high level specification.

4 Design and Specification of a Safe Control Function

Below, we give a formal definition of the safe state predicate, SS. This predicate consists of the conjunction of four safety *constraints*. A state of an object train is *safe* iff there exists at least one acceleration that allows it to simultaneously satisfy each constraint. Constraint expressions are not computable from the primitive constructs in our domain language. However, using theorems that equivalently define these comparisons in terms of discrete computations, we have developed transformations to produce an executable specification.

From the perspective of an object train, the slower it moves the safer it is; if the object train is safe with respect to a continous stopping profile that is maximum among all possible continuous profiles consistent with its given discrete stopping profile, that is a conservative approximation. In contrast, the faster the lead train moves, the less likely that the object train will collide with it. Thus, minimum among all continuous profiles consistent with a discrete stopping profile of the lead train is a conservative approximation.

Definition 4.1 $maximum(\mathcal{CP}) \stackrel{def}{=} ctp_1$: $ctp_1 \in \mathcal{CP} \wedge [\forall(p,s,t) \in ctp_1, \forall ctp_2 \in \mathcal{CP}, \forall(p_1,s_1,t_1) \in ctp_2 : (p = p_1 \wedge t = t_1) \rightarrow s \geq s_1]$

Definition 4.2 $minimum(\mathcal{CP}) \stackrel{def}{=} ctp_1$': $ctp_1 \in \mathcal{CP} \wedge [\forall(p,s,t) \in ctp_1,$ $\forall ctp_2 \in \mathcal{CP}, \forall(p_1,s_1,t_1) \in ctp_2 : (p = p_1 \wedge t = t_1) \rightarrow s \leq s_1]$

In the above definitions, if $\mathcal{CP}$ is a set of continuous profiles consistent with a given discrete train profile tp, then $maximum(\mathcal{CP})$ and $minimum(\mathcal{CP})$ exist.

Let OT and LT be discrete profiles, respectively, of an object train and its lead train. Given a discrete normal stopping profile $nsot$ of an object train in state $OT[t]$, it is safe to assume the continuous normal stopping profile, cp_{nsot}, of the object train to be $maximum(\mathcal{CP}_{nsot})$, as in reality, the continuous normal stopping profile will be based on the object train moving slower. Similarly, it can be safely assumed that the continuous normal stopping profile, cp_{slt}, corresponding to a stopping profile slt of the lead train in state $LT[t]$ be $minimum(\mathcal{CP}_{slt})$, as in reality, the continuous normal stopping profile will be based on the lead train moving faster. Maximum and minimum continuous normal stopping train profiles are, thus, conservative profiles with respect to the observed stopping behaviors of the object and lead trains.

Let $\mathcal{C}1_<$ and $\mathcal{C}2_<$ be constraints based respectively on the track profile and signals profile. Let $espot$ denote the continuous train profile corresponding to the emergency stopping profile of the object train. Let $\mathcal{C}3_<$ and $\mathcal{C}4_<$ be constraints due to the continuous normal stopping profile cp_{slt} of the lead train, and the continuous profile corresponding to a derail of the lead train.

$$SS(OT[t], LT[t], track, signals) \stackrel{def}{=} cp_{nsot} \ll \mathcal{C}1_< \wedge cp_{nsot} \ll \mathcal{C}2_< \wedge$$
$$cp_{nsot} \ll \mathcal{C}3_< \wedge espot \ll \mathcal{C}4_<.$$

The definition below states that safe transitions, ST, are transitions from one safe state to another safe state or a transition to an emergency stop in the case where the initial state is not safe.

$$ST(OT[t], OT[t+1], LT[t], track, signals) \stackrel{def}{=}$$
$$[SS(OT[t], LT[t], track, signals) \rightarrow SS(OT[t+1], LT[t+1], track, signals)] \wedge$$
$$[\neg SS(OT[t], LT[t], track, signals) \rightarrow is\_type(a') = emergency\_stop].$$

An arbitrary control function is *safe* iff it only makes safe transitions.

$$SafeController(Control) \stackrel{def}{=}$$
$$\forall(OT, LT, t, track, signals) : \ \mathrm{ST}(\ OT[t], OT[t+1],\ LT[t], track, signals),$$

where the acceleration $a' = Control(OT, LT, track, signals)$ is used to compute $OT[t+1]$ from $OT[t]$.

$Control(OT, LT, track, signals) =$
$select(\ [\mathcal{N}\_\mathcal{A}(s,a) : \ \lambda b.\ \mathrm{ST}(\ OT[t], OT[t+1],\ LT[t], track, signals)])$
where $OT[t] = (p,s,a)$ and $OT[t+1] = (p',s',a')$ and $p' = p + s$. Here the speed and acceleration are given in $\frac{1}{2}$ units.

The function $Control$ for the object train is safe if it only *selects* accelerations that result in safe transitions (ST). $\mathcal{N}\_\mathcal{A}(s,a)$ denotes the set of possible next accelerations for a train with current speed s and current acceleration a. The colon operator is a filter function (discussed further in Section 5.1) that when given a set, $\mathcal{S}$, and predicate, $\mathcal{P}$, will return the subset of $\mathcal{S}$ whose elements satisfy $\mathcal{P}$. For details, see [13].

4.1 Correctness of Formal Model

The continuous view of the system allows safe behaviors to be formally stated in a direct and concise manner. Above, we specified a control function that first generates the set of all accelerations leading to controllably safe behaviors, and then selects the maximum acceleration from the set. We would like to point out that other accelerations could be selected, such as an acceleration that minimizes *oscillation*, but such considerations are beyond the scope of our present study.

The following two theorems serve as the basis to obtain a provably correct implementation from the given specification. These theorems relate (safety) constraints expressed in a continuous domain to their discrete counterparts. Using these theorems, it is possible to transform the specification to an algorithm that can be computed in a framework based on finite enumerated sets. The proofs for both of these theorems can be found in [13].

Theorem 4.3 *If* $OT' = maximum(\mathcal{CP}_{OT})$*, where* OT *is a discrete profile of an object train, and* $B' = \mathcal{CP}_B$*, where* B *is a discrete component profile describing a track or signal configuration, then*

$$\begin{aligned} OT' \ll B'_< \Leftrightarrow\ & (\forall j,t : OT[t] = (p_1,s_1) \wedge OT[t+1] = (p_2,s_2) \wedge B[j] = (p_3,s_3) \wedge \\ & p_1 \le p_3 \le p_2 \rightarrow (\max(s_1,s_2) < s_3))) \wedge \\ & (\forall j,t : OT[t] = (p_1,s_1) \wedge OT[t+1] = (p_2,s_2) \wedge B[j] = (p_3,s_3) \wedge \\ & B[j+1] = (p_4,s_4) \wedge p_3 \le p_1 \le p_4 \rightarrow (\max(s_1,s_2) < s_3))). \end{aligned}$$

Theorem 4.4 *If* $OT' = maximum(\mathcal{CP}_{OT})$ *and* $LT' = \{minimum(\mathcal{CP}_{LT})\}$*, where* OT *and* LT *are, respectively, discrete profiles of an object train and its lead train,* $OT' \ll LT'_< \Leftrightarrow (\forall t : OT[t] = (p_1,s_1) \wedge LT[t] = (p_2,s_2) \wedge p_1 < p_2)$.

5 HATS: A Program Transformation System

HATS is a transformation system developed within the High Integrity Software (HIS) program at Sandia National Laboratories. It is freely available an can be downloaded from *http://www.sandia.gov/ast/downloads.html.*

In HATS, program transformation is realized though an extended form of term rewriting. It is a language independent system where rewriting takes place within a wide spectrum language that is defined by a context-free grammar, which forms the basis of interpreting program “strings” as terms. An abstract prettyprinter is used to translate terms back into strings.

HATS has been specifically designed for rewriting in non-confluent, non-terminating systems, such as those that are typically encountered in software development. This emphasis is reflected by a special purpose transformation language that enables sophisticated match conditions as well as transformational control to be expressed. One distinguishing feature of the HATS program transformation language is that unification is an explict operation, and transform functions are parameterized by the terms they transform. This parameterization permits the arguments to unification expressions to be calls to other transform functions (including recursive calls). This allows for a very sophisticated control over the application of transformations.

To date, HATS has been used to implement:

- An optimizer for a special class of reactive systems (of which the Production Cell [7] is a member).
- A unit resolution propositional theorem prover. Mainly, this was an experiment to test the capabilities of the control paradigm supported in HATS.
- Martell-Montannari's unification algorithm for a set of equations.
- A transformation-based class loader that generates an executable ROM image from a Java class file hierarchy, performing the symbolic resolution of the constant pool entries, link editing, and loading of each class file in the hierarchy. The class loader output will be a ROM image in a format compatible with PROM programmers (e.g.,Motorola S Records), and will support the architecture of the Sandia Secure Processor.

5.1 Example: An Optimizing Transformation

In the domain language, there is an operation denoted by the colon symbol which we refer to as *filter*. Let $\mathcal{S}$ denote an enumerated set and let $\mathcal{P}$ denote a predicate on the elements of $\mathcal{S}$, then the expression $[\mathcal{S} : \mathcal{P}]$ denotes the subset of $\mathcal{S}$ whose elements satisfy $\mathcal{P}$.

Consider the expression: $[\mathcal{S} : \mathcal{P}] \neq \emptyset$, which can be used to define the $\exists$ operator. If filter is simply treated as a library function then $[\mathcal{S} : \mathcal{P}]$ will have to be evaluated first followed by a comparison with the empty set. This evaluation sequence can be inefficient when $\mathcal{S}$ is large. However, a new function, *nonempty*, can be created by distributing the comparison with the empty set over the operational definition of filter. This results in the following:

$$\text{filter}(\mathcal{S},\mathcal{P}) = \text{if } \mathcal{P}(\text{first}(\mathcal{S})) \text{ then } \{\text{first}(\mathcal{S})\} \cup \text{filter}(\text{rest}(\mathcal{S}), \mathcal{P}) \text{ else filter}(\text{rest}(\mathcal{S}),\mathcal{P}));$$

$$\begin{aligned}\text{nonempty}(\mathcal{S},\mathcal{P}) = &\text{ if } \mathcal{S} = \emptyset \text{ then } \emptyset \neq \emptyset \text{ else if } \mathcal{P}(\text{first}(\mathcal{S})) \\ &\text{then } (\{ \text{ first}(\mathcal{S}) \} \cup \text{rest}(\mathcal{S},\mathcal{P})) \neq \emptyset \text{ else nonempty}(\text{rest}(\mathcal{S},\mathcal{P}))\end{aligned}$$

The above definition of nonempty can be simplified further:

$$\begin{aligned}\text{nonempty}(\mathcal{S},\mathcal{P}) = &\text{ if } \mathcal{S} = \emptyset \text{ then } false \text{ else} \\ &\text{if } \mathcal{P}(\text{first}(\mathcal{S})) \text{ then } true \text{ else nonempty}(\text{rest}(\mathcal{S},\mathcal{P})).\end{aligned}$$

This enables the following context-dependent optimization:

$$[\mathcal{S} : \mathcal{P}] \neq \emptyset \sqsubseteq \text{nonempty}(\mathcal{S},\mathcal{P}).$$

For a correctness proof of the optimizing transformation using *RRL*, see [11].

6 Implementation, Simulation, and Validation

We have discussed above a formal model of a train system, assumptions about the model, a high-level specification of the controller satisfying certain safety criteria while maximizing throughput, and a set of optimizing transformations to obtain an efficient implementation from the high-level specification of the controller. In order to gain confidence in the process of designing a safe controller, a combination of techniques were employed. This included

(i) proving theorems about relationship between the continuous model and the discrete model to confirm that our intuition had been captured;

(ii) using an automated reasoning tool *RRL* [4] to prove properties such as optimizing transformations to generate an efficient implementation from a high-level specification is correct;

(iii) developing a simulation of the model using the efficient implementation of the controller to develop additional confidence in the design by trying different scenarios using the simulation.

Regarding (ii) above, one such optimizing transformation was discussed in the previous subsection. Its semantic correctness as well as the correctness of other general-purpose transformations used to optimize the high-level specification of the controller are discussed in [11].

Developing a formal model and evaluating different trade-offs (e.g., whether only discrete profiles be used and safety properties be expressed using discrete profiles, how to map discrete profiles into continuous profiles) involved numerous trial and errors. A variety of tools are necessary in the design phase for analyzing such complex systems.

6.1 Optimizing Transformations Lead to a Substantial Speedup

Sim Steps $\frac{1}{2}sec.$	Track Seg.	S'_0	S_n	Track Seg.	S'_0	S_n
10	100	3.938	0.344	300	24.84	0.734
100	100	40.218	4.578	300	252.41	8.5
200	100	64.719	8.078	300	401.375	14.312
...	...	...	...	...	...	...
500	100	154.203	16.906	300	963.782	37.953

An executable implementation of the high-level specification of the abstract controller algorithm was obtained in the *HATS* system by providing an operational semantics of the simple base functions used in the modeling language. The executable specification obtained was too slow to meet the real-time requirements of the system. Optimizing transformations were then applied to generate an efficient implementation.

Many general as well as domain and problem specific optimizations have been identified. However, only a few have been implemented. The optimizations that we did implement, dramatically reduced the running time of the implementation as the results in the above table indicate. The substantial speed-up enabled us to quickly try many different scenarios to enhance our confidence in the design of the controller.

6.2 Gaining Confidence via Simulation and Graphic Display

The design of the controller has been guided by the constraint imposed on the behavior of a train by its lead train on the track. During the design phase, a major concern arose whether the system throughput was severely affected because of a overly conservative view taken above about the behavior of an object train and its lead train for calculating stopping profiles. I.e., if there are many trains on the track, consecutive trains in a sequence would travel slower and slower, with the last train not moving at all or perhaps moving very slowly. While such a property could be, in principle, be formally stated and checked using reasoning tools, it would have required a considerable effort. Simulation can be very helpful in such cases, especially when there are doubts about certain aspects of the design, and it is unclear whether substantial resources should be invested in developing a rigorous correctness analysis.

With an efficient implementation available, we quickly added a few additional transformations, allowing us to synthesize a collection of control functions for a sequence of 10 trains (it was very easy to change the number of trains running on a single track; we started with a few trains and kept on increasing their number until we felt we had a realistic model).

In order to observe the behavior of a train system, we hand-coded a simulation environment. Since the property being investigated was not going to be affected by the presence of traffic signals or stations, they were not included in the simulation. This environment generated a track consisting of track segments with associated speed limits together with an initial configuration for the trains in the system. The control function for each train computed the optimal acceleration for the next time interval. This acceleration was returned to the simulator which applied it to the corresponding train. The trains in the system were then advanced, for a $\frac{1}{2}$ second interval, according to the motion formulas given in the BART case study document, generating a new system state which was in turn fed back to the control functions. Typical sample simulation runs would create a track of a few hundred miles and would run the trains over the length of the track.

A vector of position and speed tuples for each train in the train system was generated; such a vector was checked to assure that they satisfied the following basic safety properties: (1) no train was exceeding its track speed limit, and (2) the k^{th} train was positionally behind $(k+1)^{th}$ train.

The numerical output produced a list of vectors of numbers which often made little sense. Certain patterns of numbers would appear to be intuitively wrong (e.g., $(k+1)^{th}$ train speeding up whereas k^{th} train slowing down). To verify that the simulation was indeed correct, would lead us to time-consuming manual analysis, only to reveal that the behavior was indeed consistent with the system specification. After a few of such calculations, a more effective way to view and analyze the data being produced was needed.

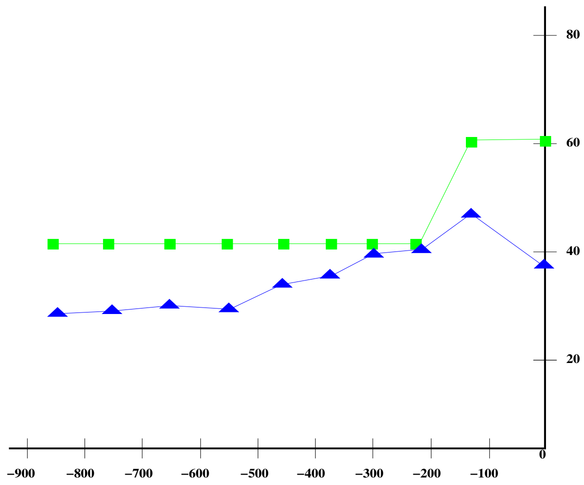

Fig. 2. Graphical Animation of 10 Trains

An obvious solution was to display the output of the simulation in graphical terms. The output was displayed in relative terms with respect to position (i.e., the position on the x-axis where the lead train is located remains fixed at the right side of the display), and the position of the remaining trains is displayed relative to the fixed position of the lead train. The speed of trains varies along the y-axis. In the animation shown, trains were represented by a blue triangle shape, and the set of trains were connected by a thin blue line. This line was added to help visualize the speed variations among the trains. Track speed limits were displayed by green squares, and were also connected by a thin green line in order to better view the relationship between train behaviors and track speed limits. Each train "triangle" was matched with a speed limit "square" in a relative manner. Thus, the green squares are always located directly above their corresponding triangle.

Such an animation leads to an increased confidence in the design since the behavior of the trains is intuitively correct. To stress the control function further, we implemented a random control function to control the lead train in the system. This function would cause the lead train to randomly speed up or slow down while staying within the track speed limit. The other trains in the system, those controlled by our formally developed control functions, caused the system to display a behavior that is most closely described by a kite tail fluttering in the wind, where the knots in the tail correspond to the triangles in the simulation. This behavior is pleasing to observe, and it is easy to informally and casually convince oneself that it is correct.

From a formal verification standpoint, such a validation via simulation is unnecessary if the underlying models are correct and no mistakes have been made in the formal development. However, the information gained from observing the simulation enhances the designer's confidence that the train behaviors and the models that they are based on seem correct with respect to the perspective provided by the simulation. Further, properties which can take considerable effort to formally disprove, can be done easily using a simulation by developing a scenario. The use of a variety of tools, including reasoning tools, testing and simulation, for gaining confidence in design and its validation cannot be over-emphasized.

Model checking can be viewed as a possible alternative to simulation for debugging a formal model, especially when the system behavior can be expressed in terms of discrete states. However, the number of states to be considerd may explode in case of 10 trains running on a single long track as well as with hundreds of simulation steps. For handling continuous properties, it is unclear, however, how model checking would have helped.

References

[1] K.A. Ghosh, B.W. Johnson, and J.A. Profeta, III. *A Distributed Safety-Critical System for Real-Time Train Control.* Proc. 21^{st} Annual Conf., IEEE Industrial Electronics Society (IECON '95), Florida, Nov. 6-10, 1995, 760-767.

[2] H. Gomaa. *Software Design Methods for Concurrent and Real-Time Systems.* Addison Wesley, 1993.

[3] C. Heitmeyer. *Using SCR to Specify Requirements of the BART Advanced Automated Train Control System.* In [14].

[4] D. Kapur and H. Zhang, "An Overview of Rewrite Rule Laboratory (RRL)," *J. of Computer and Mathematics with Applications,* 29, 2, 1995, 91-114.

[5] D. Kapur and V. Winter. *On the Construction of a Domain Language for a Class of Reactive Systems.* In [14].

[6] B. Johnson and J. H. Aylor. *Reliability and Safey Analysis of a Fault-Tolerant Controller.* IEEE Transactions on Reliability, Vol. R-35, No. 4, October 1986, 355-362.

[7] C. Lewerentz and T. Lindner. *Formal Development of Reactive Systems: Case Study Production Cell.* Lecture Notes in Computer Science Vol. 891, Springer-Verlag.

[8] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety.* Springer-Verlag, 1995.

[9] C. Morgan. *Programming from Specifications.* Prentice Hall International Series in Computer Science, 1990.

[10] V.L. Winter, R.S. Berg, and J. Ringland. *Bay Area Rapid Transit District Advance Automated Train Control System Case Study Description.* In [14].

[11] V.L. Winter, D. Kapur, and R.S. Berg. *A Refinement-based Approach to Deriving Train Controllers.* In [14].

[12] V.L. Winter. *An Overview of HATS: A Lanaugage Independent High Assurance Transformation System.* Proceedings of the IEEE Symposium on Application-Specific Systems and Software Engineering Technology (ASSET), March 24-27, 1999.

[13] V.L. Winter, D. Kapur, and R.S. Berg. *Formal Specification and Refinement of a Safe Train Control Function.* Submitted to The Computer Journal (*www.sandia.gov/AST*).

[14] V.L. Winter and S. Bhattacharya. *High Integrity Software.* Kluwer Academic Publishers, 2001. This book contains a collection of articles on the BART Case Study presented at the High Integrity Software (*HIS*) Conf., Albuquerque, NM, Nov. 1999.