

Using Abstraction in the Verification of Simulation Coercion

Xinyu Liu, Paul F. Reynolds, David C. Brogan

Department of Computing Science, University of Virginia, Charlottesville, VA 22903
{xl3t, reynolds, dbrogan}@cs.virginia.edu

Abstract

Simulation coercion concerns the adaptation of an existing simulation to meet new requirements. Interactions among course-of-action options available during coercion can become sufficiently complex that full verification of the simulation as it is adapted becomes cost-prohibitive. To address this issue we introduce two forms of abstraction, as employed in the model-checking community, to support verification of critical features of the simulation. We extend existing abstraction methods to facilitate our goals, and propose a useful abstraction method based on partial traces. As a case study, we apply our abstraction methods to the verification of a coercion of an existing simulation.

1 Introduction

Simulation reuse and composition as well as multi-resolution modeling all share a common need: the ability to adapt existing simulations to meet new requirements. The traditional method for carrying out required adaptations is manual code modification, which can be extremely time-consuming. With a rise in interest in dynamic data-driven application systems (DDDAS) [2] users are now exploring dynamic adaptation of simulations in response to real-time requirements. Clearly, methods - both static and dynamic - for efficiently adapting simulations to meet new requirements have become necessary.

Simulation coercion [1], [3] is the semi-automated adaptation of a simulation with the goal of reducing the time required to meet new requirements. Coercion is characterized by a sequence of semi-automated code transforming optimizations and manual code modifications occurring in an order determined by a subject matter expert (SME) guiding the coercion process. By manipulating flexible points [4], the optimization portion of coercion transforms a simulation to satisfy specified objectives and properties. However, optimization-based

coercion alone is not sufficient to guarantee that validity-determining properties, physical laws and user conceived constraints, are preserved. Thus, coercion-oriented verification methods become indispensable to the production of correct adapted simulations.

Complete verification of a simulation is often intractable. Simulation coercion compounds the problem. Similar issues have arisen in the software engineering and hardware design communities owing to the complexity of the systems to be verified. To address the complexity issues, we borrow from the software and hardware verification communities and introduce the use of abstraction. Abstraction has been used successfully in model checking and theorem proving to reduce the complexity of verification. Our primary contribution here is to extend its use to the verification of simulation adaptation.

In simulation coercion verification, a good abstraction will:

- reduce search space by eliminating invalid combinations of flexible point values
- reduce the number of verification test runs required to demonstrate correctness of a coerced simulation

In the following sections we present two uses of abstraction in the coercion verification process, we discuss the extension of several abstraction methods, and we propose an abstraction based on partial trace which dramatically decreases the complexity of coercion verification. Finally, we present a case study.

2 Related Work

Abstraction is a recognized method for simplifying verification. It has been used in combination with both model checking and formal theorem proving to represent properties of programs that are of greater importance to the user. We review the literature germane to our goals.

Cousot et al [5] present a framework for using

abstraction in software verification. They argue that most formal methods for reasoning about programs do not reason directly about the operational program semantics but rather about an abstract model of the semantics. They propose Abstract Interpretation, a formalization of their abstraction methods.

Two forms of abstraction have been advocated in the literature: data abstraction and control abstraction. Data abstraction has been employed to reduce an infinite or intractably large program state space into a finite abstract version. A survey of data abstraction methods used in model checking can be found in Holzmann [6].

Control abstraction creates essential models of system behavior. Hudak et al [8] present control abstraction techniques in the context of model-based verification (MBV). In hybrid I/O automaton (HIOA) systems [7], modular system decomposition and abstraction are used to reason about the system's behavior at a high level of abstraction.

Sargent [9], reviews the literature of simulation validation and verification. None of the verification techniques reviewed addresses the complexity of verifying simulation adaptation. Motivated by this issue, we build upon ideas from previously published work to extend abstraction to coercion verification.

3 Abstraction in Coercion Verification

In cases where full verification is complex, possibly intractable, abstraction can enable partial verification. Benefits of an 80/20 variety often arise: 80% of what is important to the user is verified using 20% of the effort required for full verification. Building on application of abstraction methods, verification techniques such as testing, model checking and theorem proving can then be applied. In this section we propose two novel uses of abstraction, as it has been used in the formal verification and model checking communities, for verification of simulation coercion.

3.1 Use Abstraction to Guide Optimization

Coercion involves a user-guided exploration of opportunities occurring in variations of bindings made to simulation flexible points. The exploration of possibilities for even one flexible point can be computationally prohibitive. Consider a simulation with two flexible points, where each flexible point has 100 potential bindings. A search process will need 100×100 runs of the simulation. If a simulation run consumes two minutes, the coercion will cost 20000 minutes, almost 14 days. Complexity grows

exponentially as more flexible points are added.

In practice we have found that many combinations of flexible point bindings are invalid: they violate the simulation's correctness properties. So a reasonable goal is to reduce as many combinations of flexible point bindings as possible before entering the optimization stage. Abstract models can be used to screen incorrect combinations of flexible point values. The verification time of a well-designed abstract model can be considerably less than the time to verify all possible combinations of bindings to flexible points. Furthermore, the incremental use of abstraction during coercion adds user insight to the optimization process by revealing various boundary conditions relating to correctness.

3.2 Use Abstraction to Check Coercion

The primary goal of simulation coercion is to adapt an existing simulation to meet a new set of requirements. The degree to which this goal is satisfied is usually characterized by an objective function. For example, the objective function for a physical combustion simulation might be to optimize fuel delivery velocity. However, a satisfactory solution may give a good value for burning velocity, but the simulated flame structure (e.g. temperature and chemical species spatial profiles of the fuel) may be unacceptable, e.g. the flame is too broad or various species are not properly consumed. In achieving an objective the simulation violates critical abstract constraints. Satisfaction of an objective function does not guarantee satisfaction of important properties. By capturing critical properties, we can address the need to satisfy an objective within given constraints.

As another consideration, one should be able to extrapolate the results of a coercion, to apply to cases not specifically covered in the coercion process. Coercion is typically performed based on selected cases, so the resulting coerced model may not maintain critical properties under more general circumstances. By employing an abstract verification model one can ensure the correctness of coerced models when they are used under conditions that did not arise directly during coercion.

4 Abstraction Methods

We have just argued that guiding optimization and verifying a coerced simulation are two uses for abstraction methods in support of coercion. Next we present the existing abstraction methods that can be used to implement them. For coercion, method

choice should not only address critical properties but should also take flexible points into consideration. An important part of the abstraction process will be to reduce the potential combinations of flexible point bindings likely by orders of magnitude.

4.1 Data Abstraction

Data abstraction refines the state diagram of a program by reducing possible total states. We discuss the extension of data abstraction to simulation coercion verification.

Selective Data Hiding

As traditionally used, selective data hiding removes those data objects that are not relevant to critical properties. As long as the irrelevance of removed objects can be proven, selective data hiding retains both logical soundness and completeness. For coercion verification, selective data hiding can be implemented by identifying irrelevant objects using data flow and data dependency analyses. Irrelevancy would be determined by user-guided designation of which flexible point bindings have no impact on user-specified critical properties. By removing unrelated flexible point bindings, the dimensionality of the verification space can be effectively reduced. Thus the complexity of filtering through combinations of flexible point bindings is also reduced. The process for carrying out selective data hiding can be assisted by existing program slicers [6] [7].

Data Approximation

Often, data approximation is used to map uncountable sets of data values to countable sets of values. Generally this reduces the set of values that must be considered in a verification. Data type abstraction is an example. It reduces the type of a variable to one of its subtypes. For example, floating point values are often mapped to a subset of the integers.

When verifying simulations, one is often concerned with properties associated with observable behaviors of the simulation rather than single data variables. In fact it may not be evident how to represent a given property in terms of mapping the range of values variables may take on to smaller sets. In such a case data interpolation can be used to approximate a set of data. Interpolation has been used in numerical analysis to approximate a complex function. Used as an abstraction method for verification, interpolation can be employed in cases where observable behaviors are viewed as functions over aggregations of sets of variables.

4.2 Control Abstraction

Control abstraction refines the possible control flow of a simulation by removing unused control transitions. In this section, we discuss the use of control abstraction in coercion verification.

Behavior Reduction

Behavior reduction removes uninteresting components of a simulation while preserving characteristics relevant to verification of those properties deemed important. For example, in a simulation of combustion, if only the rate of combustion is of concern, then factors such as motion of the combustion chamber can be elided.

Behavior reduction applied to coercion verification facilitates separation of concerns. Behaviors related to a property deemed important are isolated, as are relevant flexible points. Others can be ignored. Employing only isolated flexible points in analysis and verification of the abstract model, a clearer view of their impact on a particular property arises, thus reducing verification complexity. This method applies only to the optimization portion of coercion, where flexible point selection is involved. Manual code modification requires its own verification methods.

Decomposition

Decomposition is a technique for systematically partitioning a system into structural or functional components. By leveraging the simplicity and independence of components, when possible, decomposing a system and verifying each component separately can greatly simplify verification of the whole system. Decomposition can work well with federations. By decomposing a federation, each component can be verified separately, and results from individual verifications can be combined at a higher level of abstraction. Of course, one must exercise caution in such cases because a federation can take on properties not present in individual components. However, one should always be aware of the opportunity to exploit decomposition.

4.3 Partial Trace Abstraction

The methods presented in sections 4.1 and 4.2 are used often in formal verification and model checking. However, they have some shortcomings when applied to simulation coercion, including: 1) they only employ static information and are not applicable to studying behaviors resulting from simulation

execution or real system states; 2) they require execution of most of the simulation, consuming significant time and resources; 3) the abstract model they generate is subject to change during the modification and optimization process of coercion.

To address these issues, we propose an abstraction method based on partial trace, which has been employed in other verification methods. Our contribution is the consideration of partial trace for simulation coercion verification.

Partial Trace

A finite partial execution trace $s_0s_1\dots s_n$ begins in a state $s_0 \in \Sigma$ and transitions from one state $s_i, i < n$, to another s_{i+1} such that $\langle s_i, s_{i+1} \rangle \in T$. [10]

Σ is the set of possible states a simulation can enter. T is the set of possible traces. We define our partial trace-based abstraction method as follows:

1. Analyze a particular invalid coerced simulation, and extract profiles of the run which lead to violations of user required properties;
2. Compare trace results with expected results;
3. Construct a starting state s_0 from valid profiles in 2), and select trace length n long enough to let the coerced simulation manifest or violate the property;
4. Develop an abstract model represented as a Partial Trace $(s_0s_1\dots s_n)$.

Since the starting state of the abstract model resulting from step 4 is obtained from an analysis of expected results, it will not vary during the coercion process. The length of a partial trace is often very short compared to a trace for the entire simulation. This makes it an efficient abstraction method to filter invalid flexible point combinations.

5 Case Study

To demonstrate the application of abstraction techniques to simulation coercion and verification, we utilize a pair of physical simulations that have been published in previous studies of simulation coercion [11]. The two physical simulations, differing in their simulation fidelity, reproduce the movements of a human bicyclist following a target path. The coercion goal of this study seeks to make the low-resolution hockey puck, which is a one-degree-of-freedom particle-based simulation, generate the same trajectories as those generated by the high-resolution bicyclist, which uses a rigid body simulation composed of 15 controlled degrees of

freedom to model the bicyclist's limbs and bicycle components. The coercion process will manipulate two flexible points of the hockey puck simulation:

- *maximum rotation acceleration (MRA)*, the maximum degrees per second the hockey puck is allowed to change turning speeds. This variable influences the overshoot observed when bicyclists are unable to accomplish tight corners.
- *lookahead*, an internal parameter of the hockey puck's steering control algorithm. To simulate how bicyclists smooth, or "cut," corners, the hockey steering controller steers towards a position on the target path that is ahead of it by an amount proportional to the *lookahead*.

All models in this case study are based on a looping target path, shown in Figure 1. The path robustly tests the simulation behavior by presenting smooth and sharp corners that turn left and right. In the following sections, we use abstraction to expedite coercion and to validate its results.

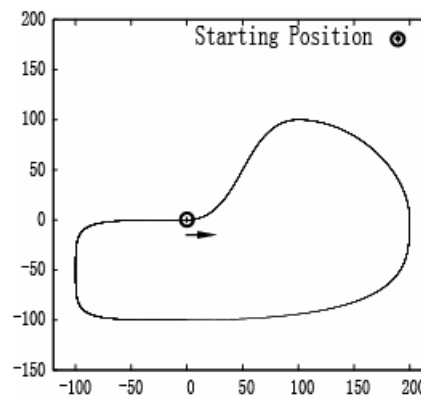


Figure 1. Target path

5.1 Abstraction for Guiding Optimization

The optimization stage of coercion requires finding the best values for the flexible points, *MRA* and *lookahead*. In this study, candidate flexible point values for *MRA* and *lookahead* are identified by the SME. According to the SME's insight, sampling of *MRA* and *lookahead* values should not be constant. There is a sweet spot that must be explored carefully. For example, for values of the *MRA* between 18 and 180, steps of size one are used, while steps of size ten are used for values between 180 and 1,800. A total of 18,821 combinations of the two flexible points are sampled. If brute-force search were to be used, days of simulation would be required to perform a hockey puck simulation trial around the entire target path for each possible pair of flexible point values.

We reduce the time required to evaluate all the flexible point values by deriving two properties that

the coerced simulation must satisfy. These properties support the overriding objective of the hockey puck simulation coercion, to duplicate the behavior of the high-resolution simulation.

Property 1

The distance between a point on the hockey puck path and the closest point on the given target path must be less than ϵ .

Property 2

The angle between the hockey puck's orientation and the path's tangent (measured at the closest point to the hockey puck) must be smaller than 90 degrees.

To a degree dictated by the high-resolution simulation, Property 1 ensures that the hockey puck is close to the target path while Property 2 ensures it does not move erratically.

Data abstract model for Property 1

The parameter sweeping required by the flexible point optimization will evaluate Property 1 at each simulation step. However, it is expensive to compute the closest point on the path for the calculation of tangent because there is no closed-form solution and an iterative procedure is used. Compounding the cost of this iterative process, the process's inner loop requires the evaluation of a third-order spline (because the target path is a cubic spline formed from eleven control points). A detailed analysis shows that calculating the closest points on the target path consumes 50% of the total simulation time. We use data abstraction to reduce the computational burden of calculating the closest point on the target path while preserving an adequate degree of accuracy. Continuity of the path is important for Property 1, but the derivatives can be discontinuous. By converting the path to a piecewise-linear curve and using linear interpolation, the computational complexity becomes little greater than the lower-resolution versions. Figures 2 and 3 show multiple discretizations of the target path and the corresponding effects on the hockey puck simulation. In both figures, the path constructed from 500 segments is so similar to the continuous spline that it is not indicated in the left pane. Figure 2 demonstrates how significant discretizations can have little effect on simulation behavior. Figure 3, however, demonstrates that although these four versions of the target paths are quite similar, they can produce dramatically different behavior.

The hockey puck simulation results reveal that behavior degrades as the target path is simplified. Because there is little cost to computing the 500 segment path, we chose to use it for the parameter sweep optimization. Additionally, we set ϵ to 20 for the parameter sweeping. It takes about 3 hours to run all combinations of different flexible point values and to verify Property 1 in each combination. Of 18,821 pairs of flexible point values tested, 2,336 tests could be preempted because Property 1 was violated. A filtering percentage of 12.4% reveals a significant reduction in the number of tests required to conclude the optimization stage of coercion.

The graphs in figure 4 demonstrate exactly where in the flexible point search space Property 1 was able to cull inadequate flexible point settings. Grey dots are valid combinations and the black ones near the origin are invalid ones. Note that the grey dots are not evenly distributed because of the heterogeneous sampling recommended by the SME. The right pane gives an enlarged picture of the corner area. It can be

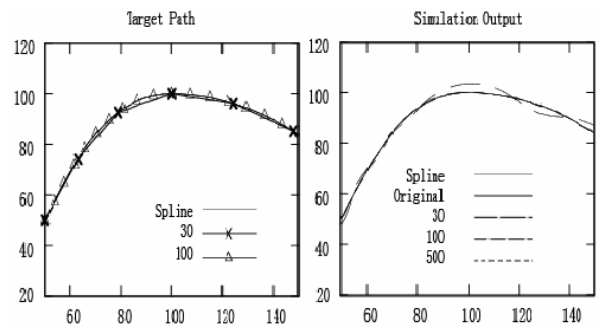


Figure 2. Path discretization (left) and the resulting hockey puck behavior (right) in a smooth curve

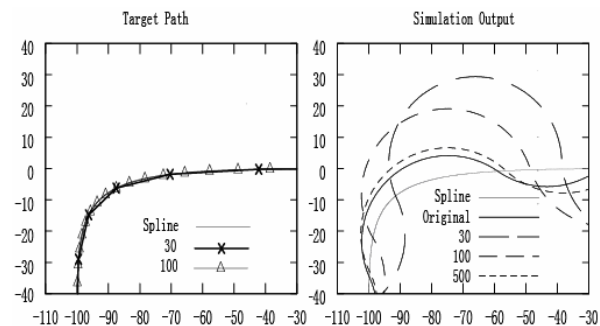


Figure 3. Path discretization (left) and the resulting hockey puck behavior (right) in a sharp curve

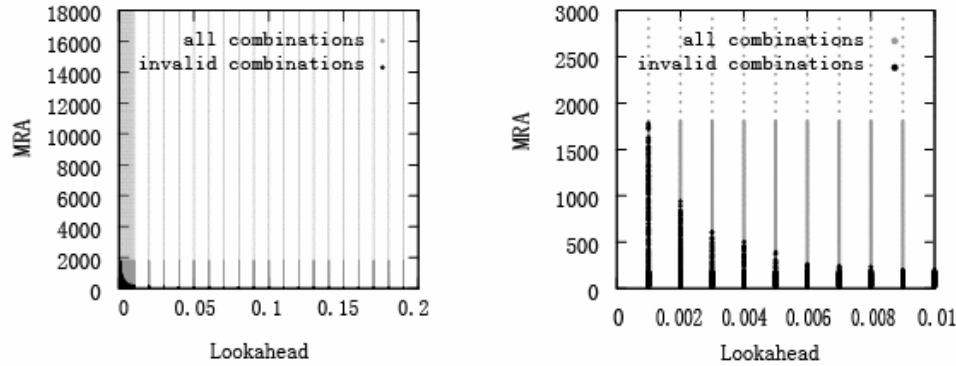


Figure 4. Results from the verification of the data abstract model

concluded that when lookahead and MRA are small, the hockey puck easily falls out of the target path. The conclusion is expected since the hockey puck, with the limitation in lookahead and MRA, cannot respond quickly to high frequency variations in the prescribed path.

Partial trace abstract model for Property 2

When searching for optimal flexible point values, the Property 2 must be satisfied throughout each simulation trial. Violation of Property 2 at any time step during the trial eliminates the candidate flexible point values from consideration. It is therefore only necessary to detect any single violation of Property 2 in simulation trials even though multiple violations may exist. This motivates the use of partial trace abstraction.

Following the steps in section 4.3, we first look for a typical invalid simulation state. Figure 5 shows the hockey puck path under the combination (lookahead=0.01, MRA=10). Taking a close look at the figure we can find that in several positions the hockey puck violates Property 2. By using the partial trace abstraction, we define a partial trace that begins before the invariant violation occurred. For example, in figure 5 the black circle at position (128.515, 59.576) serves as a valid state that precedes an invalid one. In this valid state, the hockey puck satisfies Property 2 but its direction is going to be contradictory to the target path direction in the next few steps. By initializing the simulation to perform a parameter sweep at this partial trace, we aim to eliminate flexible point values that are inappropriate for the path following requirements. The flexible point values that permit the hockey puck to successfully navigate this partial trace should be retained as candidates that will produce optimal hockey puck behavior across the entire path.

The combinations are shown in Figure 6, where grey dots are valid combinations and black dots on

the bottom are invalid ones. The right pane of figure 6 gives an enlarged picture of the lower-left corner area. We observed that when *MRA* is less than 600, the hockey puck will violate Property 2. Above this threshold, the hockey puck has sufficient rotational acceleration to allow it to adjust its direction quickly enough to be consistent with the target path direction.

5.2 Abstraction for Checking Coerced Simulation

In previous coercion studies of the hockey puck, a brute-force parameter search process was used to optimize the flexible points for a single target path [14]. A limitation of this optimization process is that only a specific target path was considered. Because the final goal is to let the hockey puck mimic the high-resolution simulation of the bicyclist as closely as possible, we are concerned with how well the resulting coerced hockey puck behaves on other target paths. We extend Property 2 to Property 3 in order to investigate how well the coerced hockey puck behaves on target paths *similar* to the one on which it was coerced.

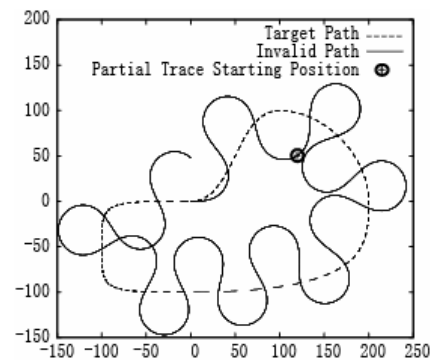


Figure 5. Invalid run of the hockey puck

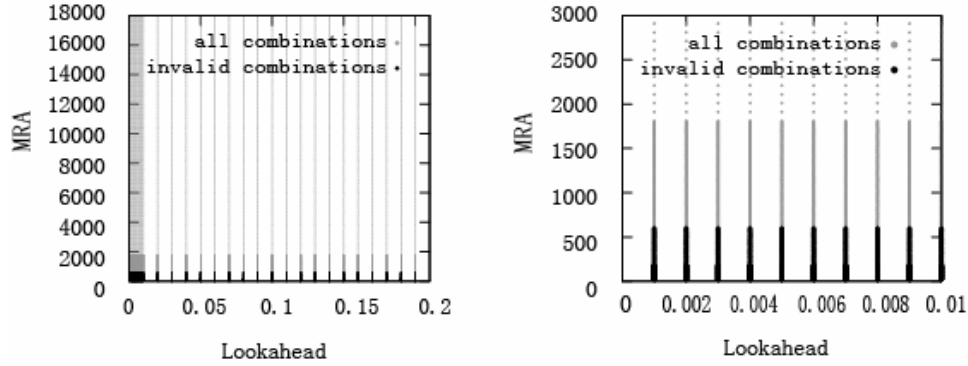


Figure 6. Results from the verification of the partial trace abstract model

Property 3

The angle between the hockey puck's orientation and the path's tangent (measured at the closest point to the hockey puck) must be smaller than 90 degrees in target paths "similar" to the one in figure 4.

Control abstract model for property 3

A steering control algorithm determines the orientation of the hockey puck at each simulation time step. We utilize control abstraction methods to investigate this algorithm. First we exploit decomposition by extracting the entire steering control procedure from the simulation, and then use behavior reduction to identify all behaviors related to the hockey puck's trajectory. Based on Property 3 and the steering control algorithm, we identify two key parameters to construct the abstract model:

- θ_1 : local curvature of the target path. We measure θ_1 by measuring the change in the target path's tangent (its second derivative) along the entire loop. For the target path shown in figure 5, the change in tangent direction, θ_1 , is always within the range of +/- 0.24 degrees per timestep.
- θ_2 : tangent direction of the target path. We measure θ_2 in the target path between the point on the path closest to the hockey puck and at the location determined by the *lookahead*. Inspection of the target path in figure 5 reveals the difference is within the range of +/- 0.85 degrees.

Given this analysis of the relevant properties of the target path as they relate to Property 3, we define *similar* target paths to be those that have curvature properties within the identified ranges:

- 1) $-0.24 \leq \theta_1 \leq 0.24$ and
- 2) $\theta_1 - 0.85 \leq \theta_2 \leq \theta_1 + 0.85$.

In the verification we create test cases that vary path curvatures and hockey puck initial conditions. We vary θ_1 by 0.001 degrees in the range of $-0.24 \leq \theta_1 \leq 0.24$.

The distance between the hockey puck and the path is varied from 0.0 to 20 meters by increments of 0.5 meters. Our careful analysis of the hockey puck's steering control algorithm reveals the hockey puck's new orientation is a linear function of θ_2 and its orientation relative to the path. Therefore, we can test for violations of Property 3 at the boundary values of θ_2 and the hockey puck's orientation. The value of θ_2 is tested at $\theta_1 - 0.85$ and $\theta_1 + 0.85$. The value of the hockey puck's orientation is tested at +/- 90.0 degrees. Using the boundary conditions and discretizations listed above, we created 76,800 test conditions to verify Property 3, and the simulation was executed for one timestep in each of these test conditions.

To evaluate the effectiveness of Property 3, we analyzed previously published coercion results to generalize to similar paths. Carnahan et al reported *MRA* and *lookahead* flexible point values of 1,580 and 0.008 respectively for a target path identical to that in figure 5 [14]. Upon testing those flexible point values for our similar paths, Property 3 was found to be violated. Figure 7 shows values for θ_1 and θ_2 that violate Property 3 when the distance between the hockey puck and the curve is 0.0 meters and the hockey puck's orientation is 90 degrees.

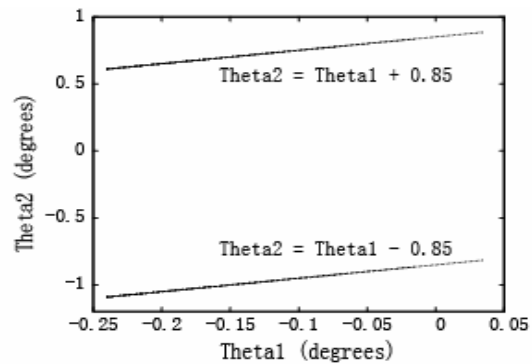


Figure 7. Invalid combinations of θ_1 and θ_2

6 Discussion

Our experiences reveal some particular strengths and limitations of each abstraction method in coercion verification. *Data abstraction* is useful in the optimization of a simulation's flexible points when they involve continuous data or data sets. However, it can produce misleading results when the selected samples fail to characterize their sets. In our case study, the discretized target path may cause hockey puck trajectories that are dramatically different from what would be produced by the continuous path.

Partial traces takes both normal and boundary cases into account like test cases. However, it integrates dynamic runtime information and only executes a few simulation steps. This fundamental difference makes rapid culling of flexible point values possible. But poor selection of partial traces can produce useless culling of flexible point values.

Control abstraction simplifies the control flow of a program and in our case study it permits us to reverse engineer the critical simulation states that must be tested to validate the behavior of the coerced simulation. Another advantage of control abstraction is its capability of dealing with large, complex simulations. Because most modern simulations follow a modular structure, the complexity of coercion verification can be significantly reduced by decomposition.

Additional abstraction can be achieved by using more than one abstraction method in sequence. In our case study, we can combine two filtering abstraction methods, e.g., a partial trace abstraction can be followed by a data approximation. Combination of abstraction methods should be used with caution because sampling errors inherent to each method can compound.

7 Conclusions And Future Work

In this paper we presented two uses of abstraction in the verification of simulation coercion: guiding optimization and checking coercion. We discussed extension of existing abstraction methods for coercion verification, and proposed an abstraction based on partial trace, which is particularly useful in coercion verification. Our case study of the coercion of an abstract bicyclist (hockey puck) simulation demonstrated the use of abstraction methods in coercion verification. Using a data approximation method, it took three hours to filter out 12.4% invalid combinations of flexible point values. Using partial trace abstraction it took five minutes to filter 31.6% invalid combinations. Then we used a control

abstraction method to check the coerced simulation. Our results show, as an unexpected but correct outcome of our analysis, that previously determined optimal bindings for flexible points in the hockey puck coercion do not extrapolate to similar but different paths for the bicyclist to follow. We interpret this finding as an excellent demonstration of the benefit of the abstraction methods we advocate in this paper.

There are additional abstraction methods that can be employed to simplify validation. For example, interval arithmetic [12] supports abstraction of real valued variables to a manageable number of potential states. We will be investigating additional abstraction methods, such as interval arithmetic, as we continue our study of ensuring the efficient demonstration of correctness during the coercion process.

References

- [1] P. F. Reynolds, "Using Space-Time Constraints to guide model interoperability," *Proc. 2002 Spring Simulation Interoperability Workshop*, Sep. 2002.
- [2] F. Darema, "Dynamic data driven application systems: A new paradigm for application simulations and measurements," *Proc. 2004 International Conf. on Computational Science*, June, 2004.
- [3] S. Waziruddin, D. C. Brogan and P. F. Reynolds, "Coercion through Optimization: A Classification of Optimization Techniques," *Proc. 2004 Fall Simulation Interoperability Workshop*, Sep. 2004.
- [4] J. C. Carnahan, P. F. Reynolds and D. C. Brogan, "Simulation-specific Characteristics and Software Reuse," *Proc. 2005 Winter Simulation Conf.*, Nov. 2005.
- [5] P. Cousot, "On Abstraction in Software Verification," *International Conf. on Computer-Aided Verification (CAV 2002)*, Copenhagen, Denmark, Jul. 2002.
- [6] G. J. Holzmann, *The SPIN Model Checker*, Addison-Wesley, 2004.
- [7] N. Lynch, R. Segala, and F. Vaandrager. Hybrid, "I/O Automata," *Technical Report: MIT-LCS-TR-827d*, MIT Lab. for Computer Science, Jan. 2003.
- [8] J. Hudak, S. Comella-Dorda, D. Gluch, G. Lewis, C. Weinstock, "Model-Based Verification: Abstraction Guidelines," *Technical Note: CMU/SEI-2002-TN-011*, Carnegie Mellon University, 2002.
- [9] R. G. Sargent, "Validation and Verification of Simulation Models," *Proc. 2004 Winter Simulation Conf.*, Nov. 2004.
- [10] P. Cousot, R. Cousot, "Basic concepts of abstract interpretation," *IFIP World Computer Congress*, 2004.
- [11] J. C. Carnahan, P. F. Reynolds, Jr., and D. C. Brogan, "An Experiment in Simulation Coercion," *Proc. 2003 Interservice/Industry Training, Simulation, and Education Conference*, Dec. 2003.
- [12] C. Muñoz and D. Lester, "Real Number Calculations and Theorem Proving," *Proc. 18th International Conference on Theorem Proving in Higher Order Logics*, 2005.