CoasterX: A Case Study in Component-Driven Hybrid Systems Proof Automation

Brandon Bohrer, Adriel Luo, Xuean Chuang, André Platzer

Carnegie Mellon University, Pittsburgh, PA 15213 USA (email: bbohrer@cs.cmu.edu, aluo@andrew.cmu.edu, xchuang@alumni.cmu.edu, aplatzer@cs.cmu.edu)

Abstract: We introduce a hybrid systems verification technique called component-driven proof automation: we exploit component structure to automate deductive verification of large-scale hybrid systems models with rich dynamics. We explore component-driven automation by implementing the first case study on formal safety verification of 2-dimensional roller coaster track designs, culminating in an automated verification tool called CoasterX. CoasterX provides a graphical front-end for point-and-click design of tracks. The CoasterX back-end automatically produces formal specifications in differential dynamic logic (dL) featuring non-linear dynamics and verifies them with a custom procedure built with the KeYmaera X theorem prover. CoasterX proves that coasters obey acceleration and velocity envelopes, showing that the forces exerted on riders are safe and the coasters do not get stuck, respectively.

Theorem-proving in dL allows CoasterX to reason exactly even under non-linear continuous dynamics and non-linear safety properties with nontrivial discrete complexity. The component-based approach overcomes scalability issues common for formal verification. We evaluate CoasterX’s practicality and scalability by proving real coasters with up to 56 components, proving in 26 minutes.

Potential future applications of our component-based approach include safety verification of road and rail networks as well as UAV flight plans.

Keywords: Roller coasters, hybrid programs, component-driven verification

1. INTRODUCTION

Theorem-proving is an effective approach for verification of hybrid systems, which are widely-used as models of safety-critical cyber-physical systems. Case studies Platzer and Quesel (2009); Mitsch et al. (2017, 2013); Loos et al. (2013); Müller et al. (2015, 2016) have shown that hybrid systems theorem-proving excels at providing a high degree of generality, e.g. supporting systems whose dynamics and safety conditions are non-linear, and provides an exceptionally high degree of assurance that the results of the safety analysis are correct.

The same case studies have also shown, however, that generality and high assurance provided by theorem proving do not come for free. Because reachability is undecidable for even simple classes of hybrid systems, most proofs of interesting safety properties are not completely automatic: users typically provide high-level verification insights through system invariants and perform low-level simplifications to assist proof automation. In practice, this limits both the scale of the models verified and the user base: interactive verification requires significant expertise.

In this paper, we show that for systems built from reusable components, it is possible to get the best of both worlds. We exploit component structure to implement reusable proof automation which verifies an entire class of systems.

Individual components are verified interactively and proof automation for verifying full models from the components is implemented by hand, both of which require theorem proving expertise. However, using the automation requires no expertise, especially because we build a graphical design tool which we connect to our proof backend, allowing truly user-friendly verification. In a well-designed theorem prover Fulton et al. (2015), this automation also does not reduce the degree of assurance: the small correctness-critical prover core detects and reports any incorrect steps attempted by automation. This is in contrast to present state-of-the-art simulation tools which, while also user-friendly, do not provide high assurance. For an analysis used to build safety-critical systems, this added assurance is an essential feature. We show that our automation scales efficiently to large models (hundreds of discrete variables), which have challenged state-of-the-art verification tools across the board.

We explore this approach of component-driven proof automation by undertaking a case study on the safety of roller coaster track designs. In addition to the first implementation of component-driven automation for hybrid systems proofs, this paper contributes for the first time a hybrid model of roller coaster dynamics and an understanding of how such a model can support practical safety questions. From a verification perspective, the model is nontrivial, featuring multi-affine differential equations and
non-linear safety regions. From a mechanical perspective, the model is simple, ignoring drag and friction. We show the surprising result that even without drag, our model displays a stark difference between safety and unsafety in real coasters.

We identify the essential safety question for coaster track designs. While roller coasters rarely collide or derail IAAPA (2017), their high accelerations not only can cause discomfort, but have been linked to life-threatening medical issues (e.g. subdural hematoma Fukutake et al. (2000)). Industry standards ASTM (2017) specify which accelerations are safe for the human body. Our proof automation verifies that these safety bounds are met with high assurance.

We take as a motivating example Kennywood’s “Steel Phantom”, which was closed and modified almost immediately Times (1991) due to excess acceleration causing neck pain, then continued to cause headaches Post-Gazette (2000) throughout its life. It was eventually closed and reopened as the gentler “The Phantom’s Revenge”, replacing over a third of the track. Design-time analysis should detect such problems early, avoiding medical risks and expensive modifications. We show that CoasterX does detect acceleration problems. For our model of Phantom’s Revenge, CoasterX automatically proved a bound of 3.55 vertical G’s, close to the 3.5 G’s of the real faker. This is much less than the 6.5 G’s for our Steel Phantom model, showing CoasterX can distinguish between safe and unsafe acceleration by a wide margin in practice.

We show that CoasterX supports a variety of designs by verifying 5 commercial coasters (Figure 1): The Steel Phantom, the Phantom’s Revenge, Lil’ Phantom, Top Thrill Dragster, and El Toro. Because commercial coaster designs are proprietary, these models are estimates. However, we also demonstrate that our components (arcs and line segments) are suitable for real coasters by verifying Gregg’s amateur coaster Gregg (2017), whose geometry has been published and matches ours exactly. We show that component-driven automation was essential for fast verification. We computed a conservative lower bound on the speedup factor, which was most pronounced on small designs (20x speedup) and significant on large designs (1.6x to 3.5x).

We posit that the component-based automation technique has the potential to provide efficient verification for other domains as well, such as flight plans for UAVs and designs for transportation systems such as rail and road networks.

2. RELATED WORK

Verification of Hybrid Systems We summarize major approaches to hybrid systems verification. Hybrid model-checkers based on reachability analysis Frehse et al. (2011) excel at automation, but few support the non-linear dynamics and safe regions needed for our models. Moreover, those which support non-linearity Chen et al. (2013) do not support modular component-based reasoning, which we crucially exploit for scalability. Model-checkers also rely on a conservative over-approximation of reachable sets, which repeatedly loses precision at every time step. This is not ideal for coasters: to maximize patron enjoyment we want maximum precision to provide the tightest possible acceleration bound.

We achieve compositional, tight reasoning via theorem proving in differential dynamic logic (dL) Platzer (2008, 2017, 2012b). Its strengths include its ability to handle system composition, non-linear dynamics, non-linear safe regions, and exact representation of reachable regions without over-approximations. All of these features are essential to the CoasterX models: non-linear dynamics are essential for realism, non-linear safe regions are needed to state meaningful acceleration bounds, and exact reachable sets improve bound precision. In CoasterX, the only inexactness is in floating-point representation of designs in the GUI tool, meaning imprecision is introduced only once and never accumulates, in contrast to reachability analysis tools. Note that dL’s ability to verify unbounded-time systems is not needed: coasters complete in bounded time.

The KeYmaera X Fulton et al. (2015) theorem prover for dL provides proof automation, but exact verification of hybrid systems is undecidable Henzinger (1996), so the automation is fundamentally incomplete and typical KeYmaera X proofs require user interaction. This is in contrast to model-checking approaches, where exactness and expressiveness of results are sacrificed in return for automation. This paper shows that by exploiting the component structure of roller coasters, we can achieve fully automatic, precise specification and verification in a theorem prover despite the complexity of track designs. Because both specification and verification are fully automatic, the resulting tool can be used without any formal methods knowledge.

Roller coasters are a class of rail systems, so our work is related to dL case studies of the ETCS Platzer and Quesel (2009) and FRA Mitsch et al. (2017) rail systems. Roller coasters have a unique focus on gravity power, so we provide the first proof with continuously-changing track grade. Roller coasters also exert unique accelerations on riders, so we prove tangential and radial acceleration bounds, in contrast to prior work.

Müller has investigated component-based theorem-proving for hybrid systems by verifying traffic network components Müller et al. (2015), using the resulting insights to build the (unverified) design tool SAFE-T. Later work Müller et al. (2017) provides a general rule for composing components in KeYmaera X and uses it for several interactive proofs. In contrast to both, CoasterX provides automated formal specification and proof for complete systems, resulting in higher assurance. SAFE-T models traffic as a (real-valued) count of cars, leading to a trivial (constant) continuous dynamics. In contrast, our dynamics are non-linear.

Distributed hybrid systems verification Platzer (2012a) also allows proof reuse for repeated components. Dis-
Coasters can get stuck Boyette (2017) during operation. Component-driven verification excels instead for systems where asymmetry is key to the safety question: e.g. a roller coaster contains many straight and curved track sections, but the exact shape of each section is essential to determining safety.

Component Modeling and Verification. Assume-guarantee reasoning Frehse et al. (2004); Henzinger et al. (2001) with Hybrid I/O Automata Lynch et al. (2003) has been proposed in model-checking to improve the scalability of verifying component-based systems when specifications are provided for the components. However, the approaches listed above only support constant and linear dynamics, respectively. Furthermore, this does not remove the need for approximation in model-checking, nor does it provide full automation because component specifications are needed.

Hybrid process algebras such as Hybrid χ Schifflers et al. (2004), the Φ-calculus Rounds and Song (2003), and HyPA Bergstra and Middelburg (2005) provide modeling frameworks for hybrid systems with components, but do not provide an approach for component-based verification, which is important for scalability.

3. ROLLER COASTER DESIGN AND SAFETY

There are an estimated 4400 roller coasters in the world Marden (2017). Because of the great forces, velocities, and heights involved, roller coasters pose inherent safety risks. Despite these inherent risks, modern coasters have a remarkable safety record: with over a billion annual rides, only an estimated 450 injuries IAAPA (2017) were reported in 2015. This safety record is achieved through pervasive safety engineering, supported by computer analysis.

Modern safety engineering uses computer-aided design (CAD) software to answer safety questions at design-time, while they can be fixed cheaply. Many design issues, including those of the Steel Phantom, reduce to analyzing acceleration envelopes. Industry standards ASTM (2017) mandate limits on acceleration on all coasters, which are different along the lateral, horizontal, and vertical dimensions. Violating these limits can have adverse medical effects Fukutake et al. (2000).

Acceleration envelopes also help determine the possibility of derailments. It is a common public misconception that trains are held to the track only by gravity (or centripetal acceleration when inverted). Modern trains have upstop wheels beneath the track which support the train during negative G’s or inversions Weisenberger (2015). Upstop wheels are strong enough to support the weight of a loaded train, making derailments extraordinarily rare. The few derailments that do occur are generally because a mechanical component failed during operation News (1986). Acceleration analysis helps determine the mechanical loads exerted on wheels and other components, providing an understanding of the risk of such failures.

Velocity analysis ensures further correctness properties. Coasters can get stuck Boyette (2017) during operation. While well-engineered rides allow passengers to be rescued without physical harm, an incident can take hours to resolve and significantly damage the operator’s reputation. Even when no physical harm results, stuck coasters are best avoided by showing a positive lower bound on velocity.

While the use of design software for coaster design is standard, existing CAD software Autodesk, Inc. (2017) is complex and has no formal correctness guarantees, making it vulnerable to implementation bugs. Inaccuracies can also result from the accumulation of floating-point arithmetic errors across many simulation steps. Additionally, verification can analyze safety of infinitely many scenarios while a simulation considers individual scenarios, e.g. we show an entire range of start velocities is safe instead of one velocity. Verification thus increases the chance that rare issues are found during design when they are easiest to fix. CoasterX provides these formal guarantees for roller coaster design with exact arithmetic and a high degree of assurance resulting from the small trusted core of the underlying prover Fulton et al. (2015).

In modeling any engineering problem, choosing an appropriate level of detail is an essential design question, with an inherent tradeoff between simplicity and accuracy. CoasterX is concerned with dynamic aspects of safety and efficiency for track designs, i.e. safe acceleration bounds and absence of stuck coasters. Thus CoasterX considers only components essential to these questions, specifically the geometry of (straight and circular) track sections in two dimensions. Other dynamical aspects, such as lateral acceleration and friction between the wheels and track, could certainly enrich our models, as would additional track shapes.

The CoasterX models are a significant advance because they enable us to prove that a real coaster (the Phantom’s Revenge) resolved its predecessor’s excess acceleration. Modeling friction is not essential to safety: while it would allow an even tighter bound, our bounds are already tight enough to draw meaningful practical conclusions. 3D coaster models with lateral acceleration would enable us to add safety results for acceleration in the lateral dimension, but Phantom’s issues stemmed from vertical acceleration in high-speed inversions, showing that vertical acceleration results are already of practical relevance. Additional track shapes are not essential because they can be approximated arbitrarily well by sufficiently many arcs. Thus, the benefit of adding such shapes would be to reduce the total number of components needed, which improves verification time. For these reasons we leave these extensions as future work.

4. TOOL: COASTERX GUI BUILDER

To maximize its audience, CoasterX seeks to provide a verification toolchain that requires no formal methods experience. This toolchain begins with allowing the user to specify track designs at a high level using a graphical design tool, instead of manually developing specifications in a formal logic. We provide this functionality through the CoasterX GUI builder (Fig. 2), implemented in Python (∼600 lines).

The CoasterX builder is completely point-and-click; Track sections are placed with a mouse click, and the builder
automatically ensures all sections are contiguous and tangent. Any unsafe sections (where the train can get stuck or go backwards) are automatically highlighted in red. For more detailed safety information, the design tool can also perform Runge-Kutta (RK4) simulation of the coaster dynamics and display an animation of the results. In order to resolve safety issues, the user can interactively adjust the coaster’s launch velocity and immediately see the impact on the model.

The use of an interactive design tool provides greatly increased usability versus manually specifying and verifying a system in the underlying logic. Because real-time detection of unsafe track sections and simulation of coaster dynamics provide immediate feedback, they enable an efficient design workflow. However, because track designs are safety-critical, it is also critical that we maintain a high degree of assurance while providing that efficient workflow. These assurances do not come from the GUI Builder.

These assurances are provided instead via formal specification in differential dynamic logic (dL) Platzer (2008, 2012b, 2017) and formal verification in the hybrid systems theorem-prover KeYmaera X Fulton et al. (2015). The CoasterX verification backend, implemented in Scala (~3500 lines), is an extension of KeYmaera X. The CoasterX backend directly imports a CoasterX model built in the GUI, constructs a formal dL specification, and automatically verifies it with KeYmaera X. Because KeYmaera X has a small trusted core (1700 lines of Scala), this provides a high level of trust in the proofs produced by the CoasterX Prover. CoasterX is the first tool to provide automated formal proofs of complete component-based systems, in contrast to prior works which either verify the components in isolation Müller et al. (2015) or provide only a composition mechanism Müller et al. (2017), but neither of which automate formal specification nor provide end-to-end proof automation.

5. BACKGROUND: DIFFERENTIAL DYNAMIC LOGIC dL

The CoasterX backend uses differential dynamic logic (dL) to formally express track designs as hybrid programs (HPs) Platzer (2008, 2012b, 2017), a program notation for hybrid systems. Hybrid programs combine basic imperative programming constructs with nondeterminism and systems of differential equations. Their syntax is given below:

\[
\begin{align*}
\alpha, \beta &::= x := \theta \mid x' = f(x) \& H \mid ?H \mid \alpha \cup \beta \mid \alpha; \beta \mid \alpha^* \\
\end{align*}
\]

where \(\theta\) is a term, \(x\) is an assignable program variable, \(f\) is a function, \(\alpha, \beta\) are HPs, and \(H\) is a formula. Assignments \(x := \theta\) discretely update \(x\) to the value of \(\theta\). ODEs \(x' = f(x) \& H\) evolve nondeterministically for any duration so long as \(H\) remains true. Assertions \(?H\) have no effect when \(H\) is true, but abort execution otherwise. Nondeterministic choices \(\alpha \cup \beta\) nondeterministically run either \(\alpha\) or \(\beta\), and sequential composition \(\alpha; \beta\) runs \(\beta\) in the state(s) resulting from \(\alpha\). Nondeterministic loops \(\alpha^*\) run \(\alpha\) repeatedly any nondeterministic number of times.

The formulas of dL consist of first-order logic operators in addition to the dynamic logic modalities \(\alpha \phi\) and \(\langle \alpha \rangle \phi\) meaning \(\phi\) holds in all or some state resulting from running \(\alpha\), respectively. All properties in this paper are \(\alpha \phi\) properties.

6. MODEL GENERATION

Once a design is created in the CoasterX GUI, the CoasterX back-end produces a formal model of the coaster in dL, then verifies it. Model generation is part of the trusted code base for CoasterX: generating a correct model is essential because a formal proof only helps if it proves the right theorem. Our models are aimed at showing bounds on acceleration, which medicine divides into horizontal (tangential) acceleration and vertical (centripetal) acceleration to assess their effect on the body.

It is essential that our models can easily express both dimensions of acceleration, but it is not essential to model every force (e.g. friction and drag) because adding them would only provide a tighter bound. By modeling only normal forces and gravity, we provide a more conservative, but safe model. Furthermore, since we consider only horizontal and vertical accelerations, we need only model the horizontal (x) and vertical (y) dimensions, not depth (z). We also assume unit mass because mass already has no impact on the net acceleration for gravity and normal forces.

CoasterX generates this model by following the component structure of the design. The dL model for each component is created by instantiating a parametric dL component model with concrete parameters from the design. The model for a complete coaster is formed by composing the component models. Components compose cleanly because each one is restricted to an evolution domain InBounds, overlapping only at their handover points as in Figure 1. For inversion-free coasters, bounding boxes \(((x_1, y_1), (x_2, y_2))\) suffice as constraints. Inversions (e.g. in Steel Phantom) introduce self-intersection and overlapping bounds. For realistic models, overlaps occur only between upward and downward sections, thus we restore a clean interface by incorporating direction of motion (Dir is dy \(\geq 0\) for up or dy \(\leq 0\) for down) in the constraint:

\[
\text{InBounds} \equiv \text{Dir} \land x_1 \leq x \leq x_2 \land y_1 \leq y \leq y_2
\]

Figure 3 illustrates the parameters for each component model.
Parametric Line Segment Model The generic model $\alpha_{L}$ for constrained nonuniform motion on the line segment from $(x_1, y_1)$ to $(x_2, y_2)$ is a linear differential equation:

$$\alpha_{L} \equiv \{ x' = v \cdot dx, y' = v \cdot dy, v' = -\frac{dy}{dx} \cdot g & \text{InBounds} \}$$

The position $(x, y)$ evolves along the direction vector $(dx, dy)$ according to the speed $v$. The speed $v$ evolves according to the resultant tangential acceleration $T$, which for constrained linear motion is the component of gravity parallel to the track $-dy/g$ (the parallel component cancels with normal force $N$). Because the train is constrained to the track, the direction of motion $(dx, dy)$ is also the track’s tangent vector. Within an individual linear section, $(dx, dy)$ are constant. The domain constraint InBounds restricts this dynamics to the bounding box of the line segment, with $dy \geq 0$ or $dy \leq 0$ depending on the slope of the track.

Parametric Arc Model We create a generic model $\alpha_{A}$ for constrained nonuniform motion on an arc by generalizing $\alpha_{L}$, with continuous evolution of the direction vector $(dx, dy)$, resulting in a nonlinear differential equation. We specify the shape of the arc with its center $(cx, cy)$, radius $r$ and bounding box $((x_1, y_1), (x_2, y_2))$. The variable $\omega$ is $1$ for counterclockwise arcs, $-1$ for clockwise:

$$\alpha_{A} \equiv \{ x' = v \cdot dx, y' = v \cdot dy, v' = -\frac{dy}{dx} \cdot g, \quad dx' = -dy \cdot \frac{\omega}{r}, dy' = dx \cdot \frac{\omega}{r} & \text{InBounds} \}$$

In forward motion, arcs above the x-axis are clockwise ($\omega = -1$) and arcs below the x-axis are counterclockwise ($\omega = 1$), vice versa when inverted. Arcs across multiple quadrants are automatically split to simplify proofs.

Parameter Instantiation To build a DC model of a concrete design, we instantiate component parameters (e.g. $cx$, $cy$, $r$ for arcs, $dx$, $dy$ for lines) with concrete values from the GUI tool. This poses a challenge because the GUI tool works with approximate arithmetic while theorems proving works with exact arithmetic.

For example, our proofs would like to assume $(dx, dy)$ is a tangent to the track (e.g. $dx = -(cy - y)/r \land dy = (cx - x)/r$ in arcs), which is rarely exactly true in floating point arithmetic. For this reason, the model generator preprocesses designs to convert from approximate floating-point arithmetic to exact real arithmetic, while ensuring that needed identities are true. This process involves rounding values to a user-specified precision. This is a form of approximation in the GUI, but it is preferable to an approximate simulation or reachability analysis because the approximation only happens once statically rather than in accumulation across many iterations, which keeps the approximations small.

Interfaces Between Components This rounding raises challenges at the interface between components. Ensuring components meet exactly at their endpoints is not a problem, but ensuring they do so with the same slope (i.e. perfectly smooth transition) is harder. While track geometries could be adapted so their slopes agree exactly, this would cause an exponential explosion in the complexity of the geometric description, making arithmetic proofs completely non-scalable. Instead, we allow slight slope disagreements between components, which can be reduced arbitrarily by increasing model precision. At the start of each component, we insert a program $\delta$ which discretely adjusts the slope to match the track exactly.

For a straight section we set slope based on the endpoints:

$$\delta_{L} \equiv (dx, dy) := \frac{(x_2 - x_1, y_2 - y_1)}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}$$

While for an arc section we set slope based on its center $(cx, cy)$, radius $r$, and direction $\omega$:

$$\delta_{A} \equiv (dx, dy) := (\omega(cy - y)/r, -\omega(cx - x)/r)$$

Composition We model the complete coaster by composing the component models. For each component $i$, we first test whether we are within its domain (InBounds), discretely adjust the direction vector ($\delta_{i}$), then follow the continuous dynamics $\alpha_{i}$. We nondeterministically choose ($\cup$) to evaluate any component that is InBounds, and repeat the process arbitrarily often ($^{*}$):

$$\alpha \equiv (?(\text{InBounds}_{1}; \delta_{1}; \alpha_{1}) \cup \cdots \cup (?(\text{InBounds}_{n}; \delta_{n}; \alpha_{n}))^{*}$$

Bounds suffice to detect the applicable components because upon entering a component, our path through it is uniquely determined by the dynamics. As we will prove, for any two adjacent components, the coaster will only ever transition between them at the one point specified in the design. When at the boundary of two sections, both sections will be InBounds, but because (as we will also prove) safe coasters maintain positive velocity $v > 0$, a safe coaster will always evolve into the latter of the two sections.

One might wonder the role of the non-deterministic loop $^{*}$. The loop is a modeling technicality, but essential to the correctness of the model. We wish to show that the coaster is safe at all points during its evolution, which we achieve by exploiting non-determinism in the model. Expressing the model as a loop is the easiest way to ensure that non-deterministic execution can end at any point within any component, thus ensuring that our safety theorem constitutes safety at every point throughout the system.
7. VERIFICATION

CoasterX verifies two classes of dL properties: (1) quantitative acceleration and velocity envelopes which ascertain safety and other correctness properties and (2) qualitative results, e.g. that components follow their expected geometries, which increase confidence in the correctness of the model. We begin with the qualitative results, which then aid in proving the quantitative results.

Conservation of Energy Any valid model should obey fundamental laws of mechanics. Because we wish to prove velocity bounds and energy is directly linked to velocity, conservation of energy is of special interest. The only energies in the system are potential energy due to altitude and kinetic energy, so we prove \( E(t) = E(0) \) at all times \( t \), where \( E(t) = KE + PE = \frac{v(t)^2}{2} + g \cdot y(t) \).

Geometric Correctness We show that both line segment and arc components satisfy the algebraic definitions of their geometry. For a line segment, we show we never leave the line \( dx \cdot (y - y_1) = dy \cdot (x - x_1) \). This proof is automatic because line segments have a simple (i.e. polynomial) solution. For arcs, we show we never leave the circle \( (x - cx)^2 + (y - cy)^2 = r^2 \). This model is non-linear and its solution lies outside decidible arithmetic, so instead of solving it we reason by differential induction Platzer (2012b). Induction shows the lemma
\[
dx = (y - cy)/r \land dy = (cx - x)/r
\]
proving that \((dx, dy)\) is tangent to the arc, then a second induction shows the result.

Velocity Envelope Given conservation of energy, velocity is a function of altitude:
\[
v(y) = \sqrt{v_0^2 + 2(y_0 - y)}
\]
where \(v_0, y_0\) are the velocity and altitude at the start of the track. This identity is only true when \(v \geq 0\), an invariant which we prove as a lemma using a differential ghost Platzer (2012b, 2017) argument based on our knowledge that initial velocity is sufficiently high. We then compute the velocity envelope by computing the extrema of \(v(y)\) across all sections \(i\) and points \((x, y)\), then prove:
\[
\min_{(i,x,y) \in \text{track}} v(y) \leq v \leq \max_{(i,x,y) \in \text{track}} v(y)
\]
for all track sections, which follows by arithmetic from the geometry of each track and energy conservation.

Acceleration Envelope We verify upper and lower bounds for tangential and radial acceleration. For both straight and arc sections, the tangential acceleration is change in speed \(v' = -dy \cdot g\). As before, we compute the envelope by checking each segment and prove:
\[
\min_{(i,x,y) \in \text{track}} -dy(i, x, y) \cdot g \leq v' \leq \max_{(i,x,y) \in \text{track}} -dy(i, x, y) \cdot g
\]
Radial acceleration is 0 in a straight section, or \(\omega_0^2\) for an arc of radius \(r\) when rotating in direction \(\omega\). Building upon the velocity computation, we derive the specification:
\[
\min_{(i,x,y) \in \text{track}} \frac{\omega_i \cdot v(y)^2}{r_i} \leq \frac{\omega \cdot v^2}{r} \leq \max_{(i,x,y) \in \text{track}} \frac{\omega_i \cdot v(y)^2}{r_i}
\]
Once we have specified the acceleration envelope, it proves by arithmetic using the track geometry and velocity bounds.

Composition We combine the above safety proofs for each component in order to verify a complete coaster. Conjoining components with non-deterministic choice \(\alpha_1 \cup \cdots \cup \alpha_n\) introduces no difficulties: To show that a choice obeys a safety property, it suffices to show the property compositionally for each branch. The change comes in introducing a loop: to show safety across arbitrarily many interactions, we must reason by loop invariant. The loop invariant \(J\) consists of a global invariant \(G\) and local invariants \(J_i\) each of which holds within the bounds of the respective segment:
\[
J \equiv G \land \bigwedge_i (\text{InBounds}_i \rightarrow J_i)
\]
For each case \(i\) of the composed coaster, the domain constraint implies \(\text{InBounds}_i\) from which we can conclude \(J_i\) holds initially on component \(i\). In each case, we show the implication \(\text{InBounds}_i \rightarrow J_j\) holds as a postcondition for every \(j\), leading to a quadratic number of cases. In principle, all but linearly-many are computationally cheap. In the case \(j = i\), we apply the component proofs from above, using arithmetic solving to prove their preconditions. In the cases \(j \neq i \pm 1\), the sections meet at exactly the handover point, and arithmetic solving suffices to show the sections agree at that point. In the cases \(|j - i| > 1\), the sections have no overlap, making \(\text{InBounds}_i\) and \(\text{InBounds}_j\) disjoint. In principle, this leads to a quick arithmetic proof by contradiction. In the present implementation, this step is not fully optimized, leading to quadratic behavior in Section 8. We expect such optimizations to be straightforward.

8. EVALUATION

CoasterX seeks to automatically prove safety for large-scale coaster models within an acceptable runtime. We show that CoasterX scales to realistic problems by modeling and proving 6 real coasters. Each coaster proves within \(\leq 30\) min, with full results in Table 1. Adding components to any proof typically improves performance. But, since this is the first verification of roller coasters, there is no other proof against which to benchmark. Instead, we give a conservative evaluation, showing the speed gain of reusing the proof of each component, which ignores, e.g. the gains from component-based arithmetic. Even this measure shows significant gains; the full benefit is almost certainly greater.

The commercial models are estimates based on publicly available materials since the exact geometries are proprietary trade secrets. To further ensure that our geometric models suffice for realistic coasters, we modeled and verified Gregg’s coaster Gregg (2017), for which exact geometry is available. We use his exact geometry.

Most of the variables in these models describe not the continuous dynamics, but the static geometry. These variable
counts exceed by far the complexity of previous hybrid systems proof efforts. For comparison, the safety proof of the ACAS X collision avoidance system Jeannin et al. (2015), one of the most complex $\Delta L$ proofs known, uses at most 27 variables even with only linear dynamics. In contrast, we achieved up to 256 variables in our examples because component-based automation makes proofs tractable by divide-and-conquer, reducing large models into many cases each of which requires a smaller number of variables.

For large coasters, a proof with component reuse is anywhere from 1.6x to 3.5x as fast as the same proof after disabling reuse. This shows the value of reuse depends greatly on the cost of the components: arcs use optimized differential invariant proofs while the line proofs use general automatic with no attempt at optimization, so straight lines dominate the runtime. One takeaway is that component reuse reduces the need to optimize components. For small coasters, the speedup is up to 20x. The speedup decreases on big models because the arithmetic proofs are quadratic. Arithmetic optimization discussed in Section 7 could make our quadratic term smaller.

Lastly, we wish to know that the bounds derived by CoasterX are tight enough to be useful in practice. While our Phantom’s Revenge model is only an estimate, we proved a bound of 3.55 vertical G’s, close to the 3.5 G’s of the real coaster. This is much less than the 6.5 G’s for our Steel Phantom coaster, showing CoasterX can distinguish between safe and unsafe acceleration in real coasters by a large margin.

9. CONCLUSION AND FUTURE WORK

In this paper, we explored the ability of component-driven proof automation to provide scalable, high-assurance verification for non-linear hybrid systems. We showed that our automation (implemented on top of the KeYmaera X theorem prover for $\Delta L$) scales well to 50+ component instances, comprising hundreds of discrete variables, in contrast to previous hybrid systems proofs. We showed that this automation, combined with user-friendly GUI design tools, provides a verification pipeline usable even without formal methods expertise. We discussed the subtleties of generating a exact specification and proof from an inexact design produced on a GUI.

To investigate component-driven proof automation, we undertook the first case study on the verification of roller coaster track designs, which are safety-critical. We identified hybrid models of roller coaster dynamics and synthesized a safety specification from industry standards: bounded acceleration. Our verification results include velocity and acceleration envelopes, thus assessing both safety and the absence of stuck coasters. We applied these analyses to several real coasters. For our motivating example of the Steel Phantom, we showed that even with apparently simple dynamics, our model is capable of distinguishing an unsafe coaster design (Steel Phantom) from a safe modification of it (Phantom’s Revenge), in this case by a wide margin of 3 G’s. Thus, design insights from CoasterX’s analyses have the potential to help designers avoid expensive coaster modifications in the future. In future work we can provide tighter acceleration bounds, faster verification times, and lateral acceleration bounds by modeling additional forces, adding new track types, and modeling in 3D, respectively. A 3D hybrid system model is conceptually analogous to the 2D model: the major change would be in the added complexity of a 3D modeling GUI.

Perhaps most importantly, the general approach of component-driven proof automation should be applicable to any verification domain featuring highly reusable components. We are interested in applying our technique to such domains in future work. For example, this approach could be used to provide formal verification of safety analyses for road and rail network CAD models, which are component-based. This technique is not restricted to designs made by an end user. It should also be applicable, e.g. to UAV flight plans produced by an automated planning algorithm, which are sufficiently large that verification is only practical if automated.

REFERENCES


