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

Rose Bohrer, Adriel Luo, Xue An Chuang, André Platzer

Computer Science Department, Carnegie Mellon University (email: rose.bohrer.cs@gmail.com, {aluo@andrew, xchuang@alumni, aplatzer@cs}.cmu.edu)

Abstract: Component-driven proof automation (CDPA) exploits component structure to automate deductive verification of large-scale hybrid systems with non-trivial continuous dynamics. We use CDPA to implement a case study *CoasterX*, which is a toolchain for designing and verifying safety of 2-dimensional roller coaster track designs. Specifically, we verify velocity and acceleration bounds. CoasterX starts with a graphical front-end for point-and-click design of tracks. The CoasterX back-end then automatically specifies and verifies the track in differential dynamic logic (dℒ) with a custom procedure built in the KeYmaera X theorem prover. We show that the CDPA approach scales, testing real coasters of up to 56 components.

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

1. INTRODUCTION

We introduce *component-driven proof automation* (CDPA), an approach for generating formal hybrid systems specifications and proofs from high-level component-based designs, in order to assure the safety of critical physical systems. We explore CDPA through our case study *CoasterX*: we present a graphical design tool and automated specification and verification backend for 2D roller coaster track designs. CDPA begins with building a high-level component-based design: in the case of CoasterX, the user builds track designs by placing components (track sections) in a click-and-point front-end GUI. The backend then automatically follows the component structure to build a formal model and verify its safety. A key advantage of this design is that the technical challenges of modeling and verification are handled entirely in the CoasterX backend: an end-user with no formal methods knowledge can design a coaster in the high-level tool and benefit from our formal guarantees.

We build our formal models in *differential dynamic logic* (dℒ) Platzer (2008, 2017), a logic for hybrid systems that supports non-linearities in both dynamics and safety conditions. The back-end deductively verifies safety using a custom procedure built on KeYmaera X Fulton et al. (2015), a theorem-prover for dℒ designed to maximize the trustworthiness of proofs.

Reachability is undecidable for even simple classes of hybrid systems, so difficult safety proofs are not automatic: users typically provide high-level verification insights through system invariants and perform low-level simplifications to assist proof automation. These manual steps demand significant expertise. Even when automatic proofs are possible, formal modeling is itself a challenge. The CoasterX study shows that for system classes described by reusable components (e.g. coaster tracks are built from track sections) CDPA can solve the specification and verification problems once and for all: We exploit component structure to implement reusable proof automation which verifies the entire class of systems. Theorem proving expertise is needed when implementing the automation and interactively verifying the individual components. However, this expertise is not needed to use the resulting system to build verified designs graphically.

The KeYmaera X prover achieves its high reliability through a smaller trusted core than competing tools and the ability to add automation without extending the core. This reliability is especially valuable when verifying safety-critical systems. We show that exploiting component structure allows CDPA to scale efficiently to models with dozens of sections without compromising reliability.

We identify safety conditions and a formal model for roller coasters. Our safety conditions come from *biodynamics*, the study of the human body's response to forces. Because deductive verification is challenging, it is essential to identify a model that is as simple as possible while providing sufficiently meaningful guarantees. Our chosen model is mechanically simple, ignoring friction, environmental effects, and lateral motion. Yet, we will argue why the insights provided by this simple model are applicable to real coasters.

We take as a motivating example Kennywood's "Steel Phantom" (Fig. 1). The Phantom caused headaches Pittsburgh Post-Gazette (2000) throughout its life, posing a risk factor for more serious medical problems Fukutake et al. (2000). It was eventually closed and reopened as the gentler "Phantom's Revenge", with over a third of the track replaced. We show that CoasterX can detect the excessive acceleration of the Steel Phantom. This design-

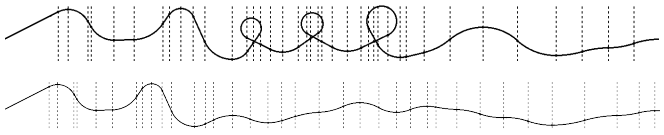

Fig. 1. Steel Phantom (top) and Revenge (bottom); dotted lines separate components

time analysis can be used to detect problems before manufacture, ameliorating medical risks and avoiding expensive changes. 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 coaster. This is much less than the 6.5 G's for our Steel Phantom model, showing CoasterX can distinguish safe vs. unsafe acceleration in practice.

We evaluate CoasterX on 6 models. We computed a conservative lower bound on the performance gain due to CDPA, ranging from 1.6x to 20x.

2. RELATED WORK

Verification of Hybrid Systems. We summarize major approaches to hybrid systems verification. Reachability analysis Frehse et al. (2011); Chen et al. (2013) provides automation, but cannot reuse verification results across component instances, which is important for performance. Its correctness typically depends on a large code base.

Instead, we use deductive verification in differential dynamic logic (d$\mathcal{L}$) Platzer (2008, 2017). It can handle component reuse, non-linear dynamics, non-linear safe regions, and exact representation of reachable sets. The CoasterX models use all of these: reuse improves performance, our dynamics are multi-affine, acceleration bounds are non-linear, and exact reachable sets improve bound precision. Because coasters are bounded-time, d$\mathcal{L}$'s ability to verify unbounded-time systems is *not* used.

The KeYmaera X Fulton et al. (2015) theorem prover for d$\mathcal{L}$ provides proof automation, which is fundamentally incomplete because reachability of hybrid systems is undecidable Henzinger (1996). Thus, typical d$\mathcal{L}$ proofs require more user interaction than does reachability analysis, gaining exact, expressive results in return. CoasterX shows that such results can be fully automated for coasters by reducing track verification to component verification.

Rail systems have been verified in d$\mathcal{L}$, including one with pressure brakes that take time to apply and which depend on the train's mass and length Mitsch et al. (2017). Roller coasters are rail systems with a unique focus on gravity power, for which we introduce continuously-changing track grade. Roller coasters exert unique accelerations on riders, so we prove the first biodynamic safe acceleration bounds.

Component-based hybrid systems theorem proving has been explored for traffic network components Müller et al. (2015), culminating in the (unverified) design tool SAFE-T. Müller et al. (2017) provide a general rule for composing components, but do not integrate it with SAFE-T. CDPA provides the first integrated approach. CoasterX shows that integrating high-level design with proof is itself non-trivial, but possible.

Distributed hybrid systems verification Platzer (2012a) also allows proof reuse for repeated components. It excels for highly symmetric systems, e.g. many cars all using the same controller. CDPA excels when asymmetries are safety-critical: e.g. our straight track component is reused many times, but *each exact track shape is safety-critical.*

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 to help hybrid model checking scale. However, it does not provide reuse across instances, nor is it fully automatic because component specifications are needed. Hybrid process algebras such as Hybrid χ Schiffelers et al. (2004) and Hybrid CSP Liu et al. (2010) provide component modeling, but component-based verification with proof reuse across instances is an open problem.

3. ROLLER COASTER DESIGN AND SAFETY

There are an estimated 4600 roller coasters in the world Marden (2018). Because of the great forces, velocities, and heights involved, roller coasters pose inherent safety risks. Despite the 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 find problems at design-time, while they can be fixed cheaply. Industry standards ASTM (2017) mandate acceleration limits for all coasters along each dimension of motion, capturing the human body's biodynamic limits. Violating these limits can have adverse medical effects Fukutake et al. (2000). These limits also help determine the possibility of derailments: derailment typically occurs if a mechanical component fails during operation CBC News (1986), and acceleration bounds help us understand whether forces between mechanical components are dangerously high.

In addition, showing a positive lower bound on *velocity* ensures coasters do not get stuck Boyette (2017) or roll back during operation, an essential correctness property. Compared with standard CAD software, soundness of CoasterX depends on only a small amount of code, due to the small (1700 lines) trusted core of the underlying prover Fulton et al. (2015). Additional benefits include the ability to consider infinitely many scenarios (e.g. infinitely many starting velocities) at once, and high confidence provided by the use of exact arithmetic in the verification.

To make deductive verification in d$\mathcal{L}$ easier, we simplify the model in multiple ways, ignoring friction, environmental effects, and lateral motion. We look to the Steel Phantom example to justify these simplifications. Our main result is an upper bound on acceleration, which is only made more conservative (and still sound) by ignoring friction. This conservative bound provides practical information: CoasterX shows a wide gap between the Steel Phantom and Phantom's Revenge. Because Steel Phantom's excessive acceleration was not along the lateral dimension, providing horizontal and vertical bounds is useful. We also

use simple components (straight lines and arcs), but these can approximate arbitrary curves at the cost of increased verification time. We model unit mass because mass cancels out of our acceleration equations.

Compared to acceleration *upper* bounds, velocity *lower* bounds help show that coasters do not get stuck or roll back. Friction and environmental (e.g. wind) effects are essential in establishing a lower bound, so velocity lower bounds should be taken with a grain of salt.

Nonetheless, the verification enabled by these simplifications allows us to draw practical conclusions, e.g. showing that the Phantom's Revenge resolved the excess acceleration of the Steel Phantom.

4. TOOL: COASTERX GUI BUILDER

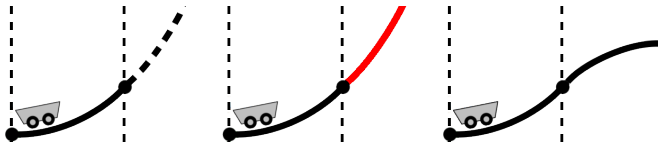

Fig. 2. CoasterX GUI: Placing track (left), Design bug detected (center), Design bug fixed (right)

The user-facing part of CoasterX is a high-level track builder GUI (Fig. 2), implemented in Python (≈600 lines).

The 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 sections where the train can get stuck or go backwards are automatically highlighted in red. The design tool can also perform Runge-Kutta (RK4) simulation of the coaster dynamics and animate the results. The user can interactively adjust the coaster's launch velocity and immediately see the impact on the safety of model.

The GUI design tool greatly reduces the learning curve vs. manually specifying and verifying a system in dℒ. Because real-time detection of unsafe track sections and dynamical simulation 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.

Our assurances come from formal specification and verification in dℒ. The CoasterX back-end extends the KeYmaera X Fulton et al. (2015) prover with custom automatic specification (≈2000 lines) and proof (≈1500 lines) procedures in Scala. Because only the specifier and KeYmaera X's core (together, 3700 lines) must be trusted, the resulting proofs have a high level of assurance.

5. BACKGROUND: DIFFERENTIAL DYNAMIC LOGIC dℒ

The CoasterX back-end uses differential dynamic logic (dℒ) to formally express track designs as *hybrid programs* (HPs) Platzer (2008, 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:

$$\alpha, \beta \ ::= \ x := \theta \mid x' = f(x) \,\&\, Q \mid ?Q \mid \alpha \cup \beta \mid \alpha; \beta \mid \alpha^*$$

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

The formulas of dℒ comprise first-order logic operators in addition to the dynamic logic operators $[\alpha]\phi$ and $\langle\alpha\rangle\phi$ meaning ϕ holds in all or some state resulting from running α, respectively. All properties in this paper are $[\alpha]\phi$ properties. See Platzer (2008, 2017) for details.

6. MODEL GENERATION

Once a design is created in the CoasterX GUI, the CoasterX back-end automatically produces a formal model of the coaster and a proof in dℒ. Our models are aimed at showing bounds on acceleration, divided into horizontal (tangential) acceleration and vertical (centripetal) acceleration per international standards ASTM (2017).

CoasterX generates this model by following the component structure of the design. The dℒ model for each component is created by instantiating a generic 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 Fig. 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:

$$\mathit{InBounds} \stackrel{\text{def}}{\equiv} \mathit{Dir} \ \wedge \ x_1 \leq x \leq x_2 \ \wedge \ y_1 \leq y \leq y_2$$

Fig. 3 illustrates component model parameters.

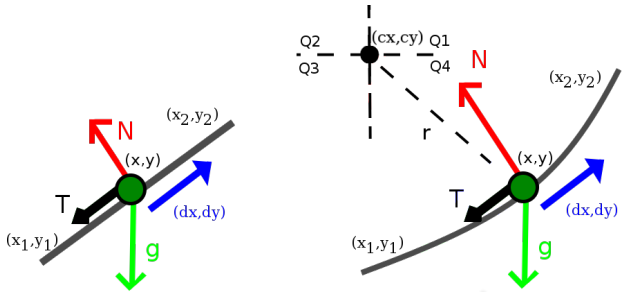

Fig. 3. Line and arc components with tangential, centripetal, radial, and gravitational acceleration

Parametric Line Segment Model. The generic model α_L for constrained nonuniform motion on the line segment from (x_1, y_1) to (x_2, y_2) is a linear differential equation:

$$\alpha_L \stackrel{\text{def}}{\equiv} \{x' = v \cdot dx, y' = v \cdot dy, v' = -dy \cdot g \,\&\, \mathit{InBounds}\}$$

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 $-dy \cdot g$ (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 any single linear section, (dx, dy) is constant. The domain constraint *InBounds* implements the segment's bounding box as described above.

Parametric Arc Model. We create a generic model α_A for constrained nonuniform motion on an arc by generalizing α_L with continuous evolution of the direction vector (dx, dy), resulting in a multi-affine 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 ω is 1 for counterclockwise arcs, -1 for clockwise:

$$\alpha_A \stackrel{\text{def}}{\equiv} \{x' = v \cdot dx,\ y' = v \cdot dy,\ v' = -dy \cdot g,\\ dx' = -dy \cdot v \cdot \omega/r,\ dy' = dx \cdot v \cdot \omega/r \ \&\ \mathit{InBounds}\}$$

Arcs above the x-axis are clockwise ($\omega = -1$) and arcs below the x-axis are counterclockwise ($\omega = 1$) normally, or vice versa when inverted. Long arcs are automatically split at quadrant (Q1–Q4) boundaries to simplify proofs.

Parameter Instantiation. To build a $\mathsf{d}\mathcal{L}$ 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. The challenge here is that the GUI tool works with approximate arithmetic while theorem-proving works with exact arithmetic.

For example, we wish for (dx, dy) to be tangent to the track (e.g. $dx = -(cy - y)/r \ \wedge \ dy = (cx - x)/r$ in arcs), which is rarely exactly true in floating point arithmetic. For this reason, the model generator pre-processes designs to convert from approximate floating-point arithmetic to exact real arithmetic while preserving properties such as tangency of (dx, dy). We round track geometry to an arbitrarily-high user-specified precision.

Interfacing 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 δ which discretely adjusts the slope to match the track exactly.

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

$$\delta_L \stackrel{\text{def}}{\equiv} (dx, dy) := \frac{(x_2 - x_1, y_2 - y_1)}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}$$

While for an arc section we set slope based on its center (cx_i, cy_i), radius r_i, and direction ω:

$$\delta_A \stackrel{\text{def}}{\equiv} (dx, dy) := (\omega(cy_i - y)/r_i, -\omega(cx_i - x)/r_i)$$

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

$$\alpha \equiv \big((?\mathit{InBounds}_1; \delta_1; \alpha_1) \cup \cdots \cup (?\mathit{InBounds}_n; \delta_n; \alpha_n)\big)^*$$

At each iteration, the coaster evolves into any component whose bounds are satisfied. For a safe launch speed, the velocity is positive and the train can only move in the rightmost such section. For an unsafe launch speed, the train can roll back to the left. By proving positive velocity, we will prove the train proceeds to the right.

7. VERIFICATION

CoasterX verifies two classes of $\mathsf{d}\mathcal{L}$ 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 always remain on the circle $(x - cx)^2 + (y - cy)^2 = r^2$. This model is multi-affine and its solution lies outside decidable arithmetic, so instead of solving it we reason by *differential induction* Platzer (2012b). Induction shows the lemma

$$dx = (y - cy)/r \wedge dy = (cx - x)/r$$

proving that (dx, dy) is tangent to the arc, then a second induction shows the train stays on the circle.

Velocity Envelope. Using the conservation of energy invariant, 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) 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:

$$\min_{(x,y)\in\text{track}} v(y) \leq v \leq \max_{(x,y)\in\text{track}} v(y)$$

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:

$$\min_{(i,x,y)\in\text{track}} -dy(i,x,y)\cdot g \leq -dy\cdot g \leq \max_{(i,x,y)\in\text{track}} -dy(i,x,y)\cdot g$$

Radial acceleration is 0 in a straight section, or $\frac{\omega\cdot v^2}{r}$ for an arc of radius r when rotating in direction ω. 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 the acceleration envelope is specified, it proves by arithmetic using the track geometry and velocity bounds.

Composition. The proofs for α_L and α_A are generic and need only be done once. We then verify a complete coaster by instantiating them to sections $\alpha_1, \ldots, \alpha_n$, verifying each branch, and conjoining them with nondeterministic choice $\alpha_1 \cup \cdots \cup \alpha_n$. We use a loop invariant to show safety holds for arbitrarily many iterations. The invariant J consists of a global invariant G and local invariants J_i each of which holds within the bounds of the respective segment:

$$J \equiv G \wedge \bigwedge_i (\mathit{InBounds}_i \rightarrow J_i)$$

For each case i of the composed coaster, the domain constraint implies $\mathit{InBounds}_i$ from which we can conclude J_i holds initially on component i. In each case, we show the implication $\mathit{InBounds}_k \rightarrow J_k$ holds as a postcondition for every k, leading to a quadratic number of cases. In principle, all but linearly-many are computationally cheap. In the case $k = i$, we apply the component proofs from above, using arithmetic solving to prove their preconditions. In the cases $k = i \pm 1$, the sections meet at exactly the handover point, and arithmetic solving suffices to show the sections agree at that point. In the cases $|k - i| > 1$, the sections have no overlap, making $\mathit{InBounds}_i$ and $\mathit{InBounds}_k$ 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

Our goal is to prove safety of large coaster models automatically and scalably. We show that CoasterX scales to realistic problems by modeling and proving 5 commercial coasters and one hobbyist coaster. The models of commercial coasters are estimated from publicly available materials since the exact geometries are proprietary trade secrets. We supplement them with a model of Gregg's hobbyist coaster Gregg (2018), which matches his published geometry exactly. Each coaster proves within ≤30 minutes on a modern workstation, with full results in Tab. 1.

There is no other coaster verification work for us to compare with, so we instead assess the speed gained by verifying the generic components once and reusing their proofs vs. verifying each instance separately. This is conservative, ignoring, e.g. the gains from component-based arithmetic; the full benefit is almost certainly greater.

The continuous dynamics are of modest complexity: they are modeled with multi-affine differential equations of at most 5 variables. At the same time, our complex geometries result in models whose sheer size significantly exceeds previous efforts in KeYmaera X, showing why our automation and component reuse were important. Our largest geometries featured 256 modeling variables and ≈50KB model files, nearly an order of magnitude more variables than previous KeYmaera X efforts Jeannin et al. (2017). Note most of the modeling variables are used to keep the length of model files manageable, and should not be confused with the continuous 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 that the value of reuse depends greatly on the cost of the components: the benefit is modest for arcs using optimized differential invariant proofs, but major for line proofs using general automation with no attempt at optimization, which dominate the runtime. For small coasters, the speedup is up to 20x. The speedup presently decreases on big models because the current, unoptimized arithmetic proofs are quadratic.

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 well between safe and unsafe acceleration in real coasters.

9. CONCLUSION AND FUTURE WORK

We introduced CDPA and implemented a case study, CoasterX, implemented on top of KeYmaera X. By exploiting component structure and reuse, CoasterX scales well to 50+ component instances. Because formal specifications are automatically generated from high-level graphical designs, CoasterX end-users need no formal methods experience. The conversion from inexact high-level models to mathematically exact specifications was subtle.

We modeled coaster dynamics and identified bounded acceleration as our safety specification. In the process, we prove velocity bounds, which assess also the absence of stuck coasters. We applied these analyses to several real coasters. For our motivating Steel Phantom example, we show that even simple dynamics can distinguish an unsafe coaster design (Steel Phantom) from a safe version of it (Phantom's Revenge), in this case by a wide margin of 3 G's. In future work we can get tighter acceleration bounds, faster verification times, and lateral acceleration bounds

Coaster Model	Sections	Geo.	Time	No Reuse	Speedup	Steps	Size
Top Thrill Dragster	8	37	23s	466s	20x	5K	4.7KB
Paul Gregg BYRC	12	57	47s	502s	11x	8K	9.2KB
Lil' Phantom	22	107	206s	708s	3.4x	14K	21.4KB
Phantom's Revenge	42	192	723s	2058s	2.8x	19K	37.9KB
El Toro	53	256	1517s	2467s	1.6x	29K	52.3KB
Steel Phantom	56	232	1120s	3965s	3.5x	26K	40.4KB

Component	Dim. (ODE)	Time	# Steps
Line	21 (3)	140s	900K
Q1 Arc	29 (5)	3.1s	9K
Q2 Arc	29 (5)	5.1s	14K
Q3 Arc	29 (5)	3.6s	10K
Q4 Arc	29 (5)	6.3s	17K

Table 1. Coaster and component proof stats. Columns (coasters): Number of component instances, # geometric variables, time with reuse, time without reuse, speedup factor with reuse, total atomic proof steps, size of theorem statement. Columns (components): total dimension (continuous vars), proof time, total atomic proof steps

by modeling additional forces, adding new track types, and modeling in 3D, respectively. These forces would improve the velocity lower bound and thus the analysis of stuck coasters. 3D vs. 2D models are conceptually similar, but a 3D GUI is a larger implementation effort.

Our results for coasters suggest CDPA is a promising approach for scalable, trustworthy, automation in other domains including rail networks, road networks, and UAV flight plans, which we wish to verify in future work.

10. ACKNOWLEDGEMENTS

Special thanks to Jim McCann, Nick Weisenberger, Brian Ondrey, Jessica Hodgins, and Hunter Lawrence for their insights about roller coasters. This material is based upon work supported by the National Science Foundation under NSF CAREER Award CNS-1054246. The first author was supported by the Department of Defense through the National Defense Science & Engineering Graduate Fellowship Program.

REFERENCES

ASTM (2017). Standard Practice for Design of Amusement Rides and Devices. Standard, ASTM Intl.

Boyette, C. (2017). 24 stuck on Six Flags roller coaster in Maryland. URL `http://www.cnn.com/2017/04/13/us/maryland-six-flags-roller-coaster-riders-stuck/index.html`. Accessed 2018-4-25.

CBC News (1986). Roller-coaster derailment kills 3 in Edmonton. URL `http://www.cbc.ca/news/canada/edmonton/june-14-1986-roller-coaster-derailment-kills-3-in-edmonton-1.3639209`.

Chen, X., Ábrahám, E., and Sankaranarayanan, S. (2013). *Flow*: An Analyzer for Non-linear Hybrid Systems*, volume 8044 of *LNCS*.

Frehse, G., Guernic, C.L., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., and Maler, O. (2011). SpaceEx: Scalable verification of hybrid systems. In *CAV 2011*, volume 6806 of *LNCS*.

Frehse, G., Han, Z., and Krogh, B. (2004). Assume-guarantee reasoning for hybrid I/O-automata by over-approximation of continuous interaction. In *CDC 2004*.

Fukutake, T., Mine, S., Yamakami, I., Yamaura, A., and Hattori, T. (2000). Roller coaster headache and subdural hematoma. *Neurology*.

Fulton, N., Mitsch, S., Quesel, J.D., Völp, M., and Platzer, A. (2015). KeYmaera X: An axiomatic tactical theorem prover for hybrid systems. volume 9195 of *LNCS*.

Gregg, P. (2018). *Backyard Roller Coasters: Research and Development*. URL `http://backyardrollercoasters.org/`.

Henzinger, T.A. (1996). The theory of hybrid automata. In *LICS*. IEEE. doi:10.1109/LICS.1996.561342.

Henzinger, T.A., Minea, M., and Prabhu, V. (2001). *Assume-Guarantee Reasoning for Hierarchical Hybrid Systems*. doi:10.1007/3-540-45351-2_24.

IAAPA (2017). IAAPA 2015 Incident Survey Report. URL `http://www.iaapa.org/intproject/download/2015RideIncidentSurveyReport.pdf`.

Jeannin, J., Ghorbal, K., Kouskoulas, Y., Schmidt, A., Gardner, R., Mitsch, S., and Platzer, A. (2017). A formally verified hybrid system for safe advisories in the next-generation airborne collision avoidance system. *STTT*. doi:10.1007/s10009-016-0434-1.

Liu, J., Lv, J., Quan, Z., Zhan, N., Zhao, H., Zhou, C., and Zou, L. (2010). A calculus for hybrid CSP. In *APLAS 2010*, volume 6461 of *LNCS*.

Lynch, N., Segala, R., and Vaandrager, F. (2003). Hybrid I/O automata. *Information and Computation*, 185(1).

Marden, D. (2018). Roller Coaster Database Census. URL `https://rcdb.com/census.htm`. Accessed 2018-4-25.

Mitsch, S., Gario, M., Budnik, C.J., Golm, M., and Platzer, A. (2017). Formal verification of train control with air pressure brakes. volume 10598 of *LNCS*.

Müller, A., Mitsch, S., and Platzer, A. (2015). Verified traffic networks: Component-based verification of cyber-physical flow systems. In *International Conference on Intelligent Transportation Systems*.

Müller, A., Mitsch, S., Retschitzegger, W., Schwinger, W., and Platzer, A. (2017). Change and delay contracts for hybrid system component verification. In *FASE 2017*. doi:10.1007/978-3-662-54494-5_8.

Pittsburgh Post-Gazette (2000). Kennywood not letting the phantom steal away. C–1,C–10. Aug 11 issue.

Platzer, A. (2008). Differential dynamic logic for hybrid systems. *J. Autom. Reas.*, 41(2), 143–189.

Platzer, A. (2012a). A complete axiomatization of quantified differential dynamic logic for distributed hybrid systems. *LMCS*. doi:10.2168/LMCS-8(4:17)2012.

Platzer, A. (2012b). Logics of dynamical systems. In *LICS 2012*. IEEE. doi:10.1109/LICS.2012.13.

Platzer, A. (2017). A complete uniform substitution calculus for differential dynamic logic. *J. Autom. Reas.*, 59(2), 219–265.

Schiffelers, R.R.H., van Beek, D.A., Man, K.L., Reniers, M.A., and Rooda, J.E. (2004). *Formal Semantics of Hybrid Chi*, volume 2791 of *LNCS*. doi:10.1007/978-3-540-40903-8_12.