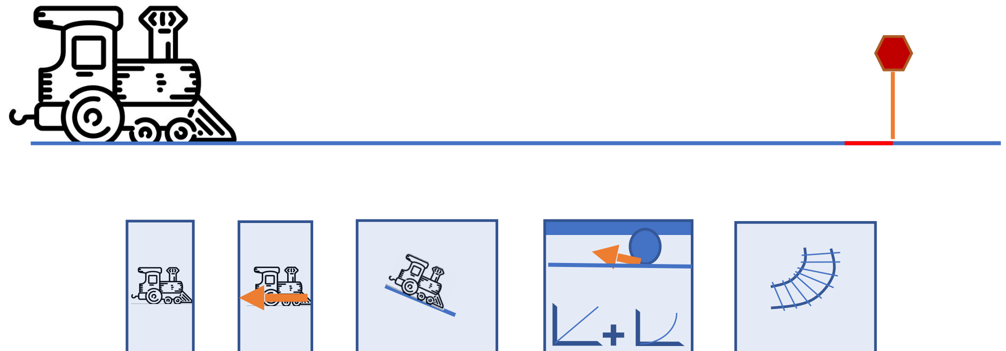
Train Verification and Control Envelope Synthesis

Aditi Kabra

Computer Science Department,
Carnegie Mellon University

KIT 06/2023

Cyber Physical Systems



$p' = v, v' = a_l + a_a + a_s(p) + a_r(v) + a_c(p), a_b' = m_b$

with $a_l \in [-b_{\max}, a_{\max}], a_a = max(a_b, a_{b\max})$

Overview

Part 1

Design Verified Train Protection System Case Study

Techniques

Results

Human Effort Intensive

Towards

Generalization
Automation

Part 2

Control Envelope Synthesis

Theoretical Characterization by Hybrid Games

Solution Computation by Refinement

Benchmarks

Pt 1: Verified Train Controllers for the Federal Railroad Administration Train Kinematics Model:

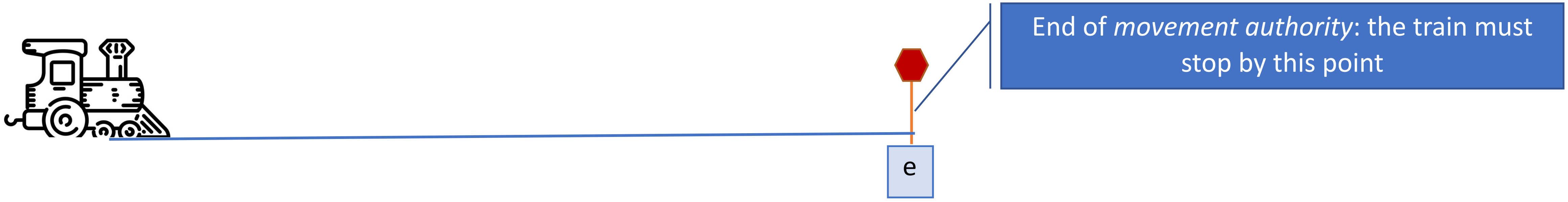
Balancing Competing Brake and Track Forces

Aditi Kabra Stefan Mitsch André Platzer

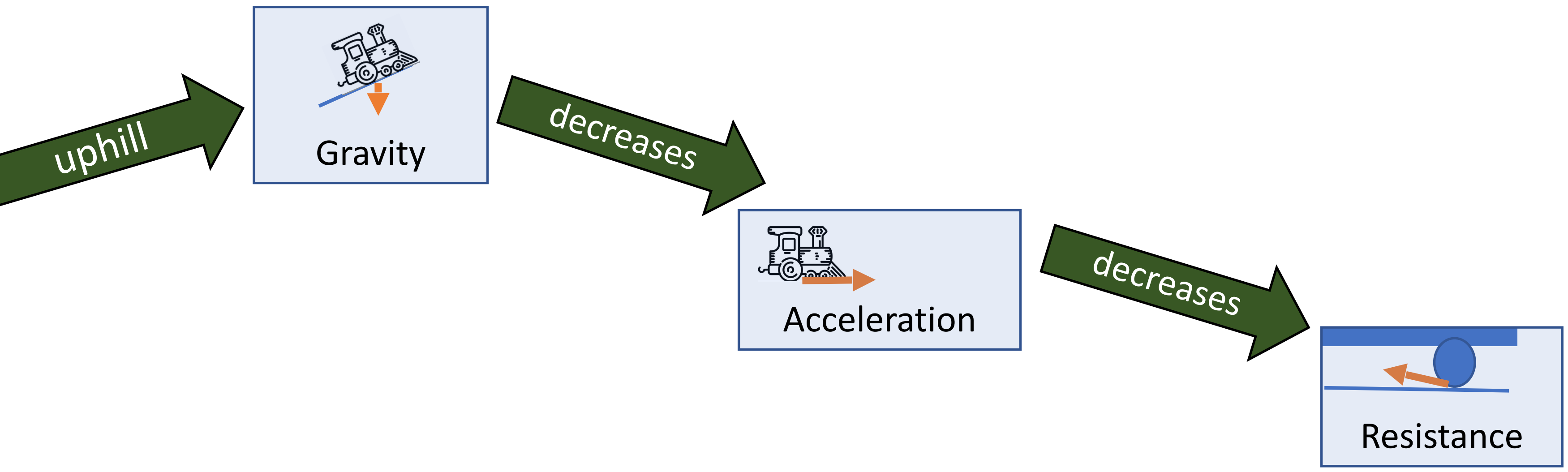
EMSOFT 2022

Supported by FRA contract number 693JJ620C000025

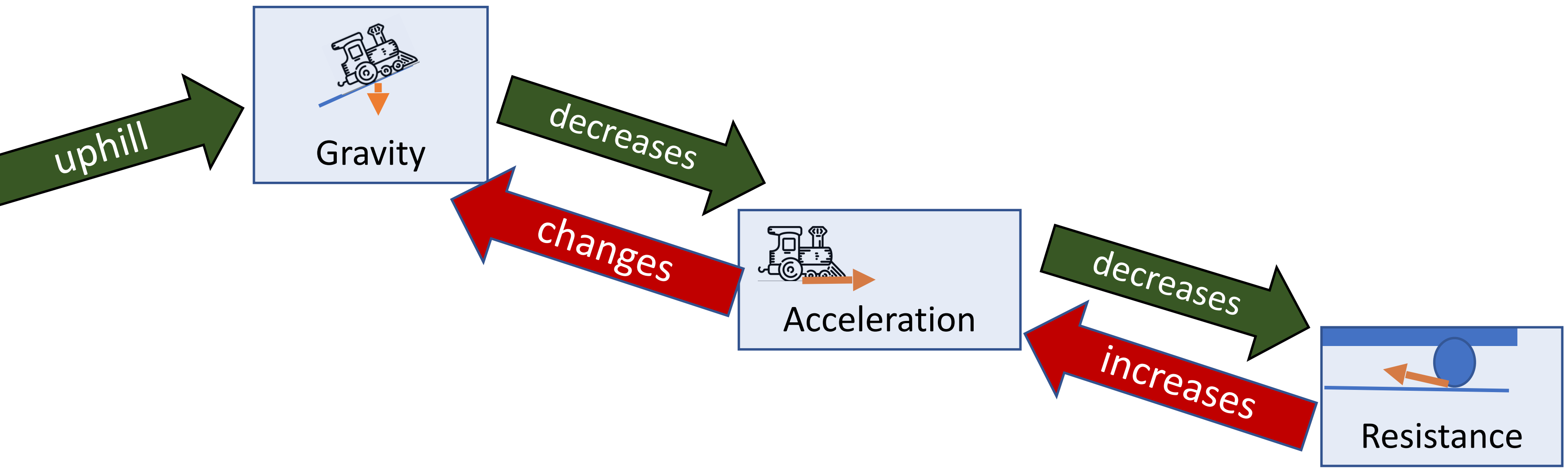
Train Control: Complicated



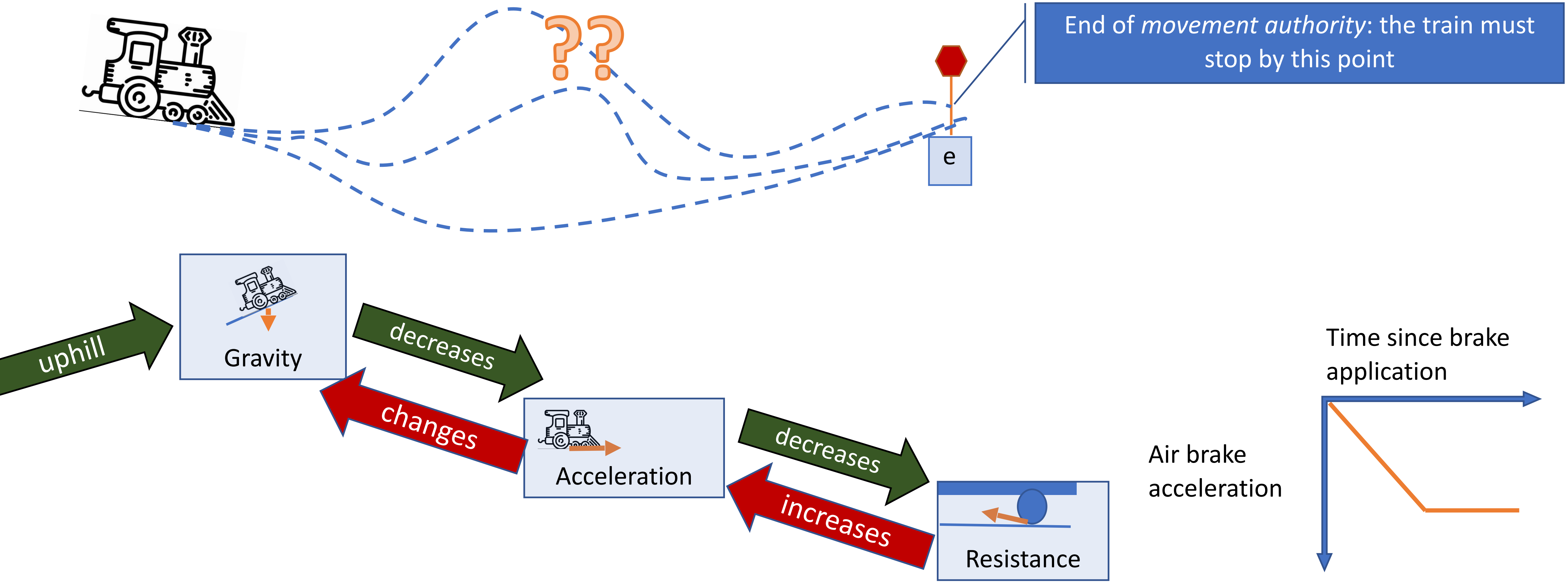
Train Control: Complicated



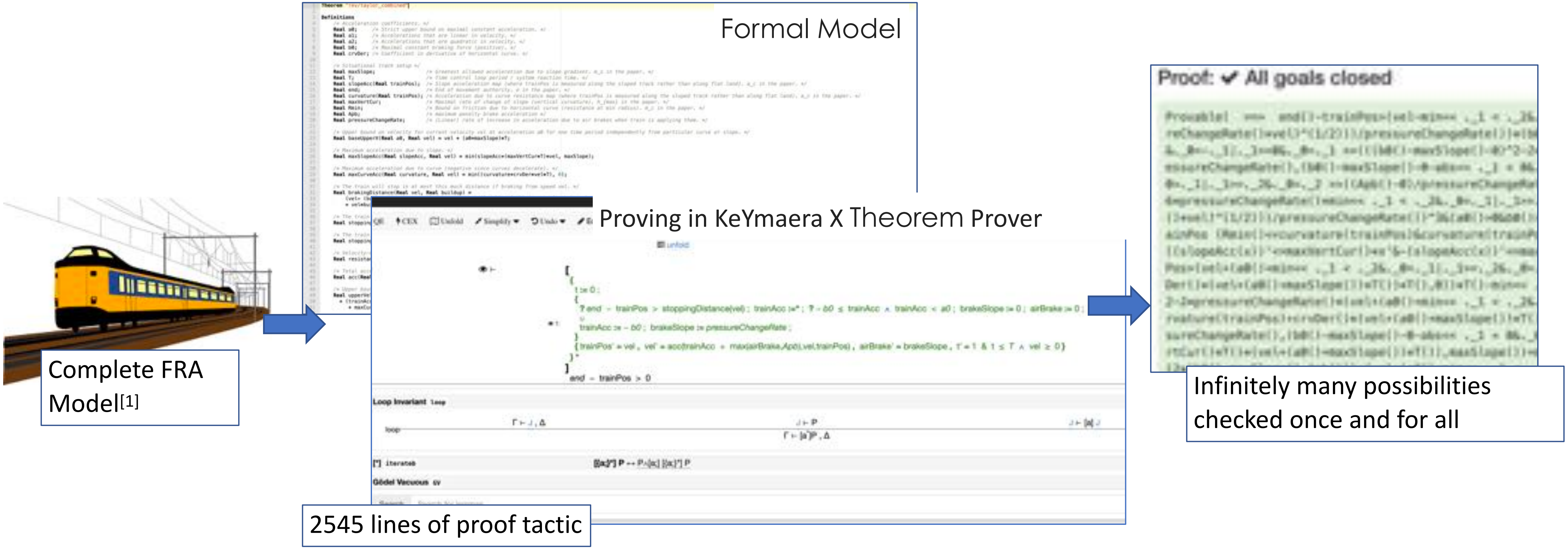
Train Control: Complicated



Train Control: Complicated



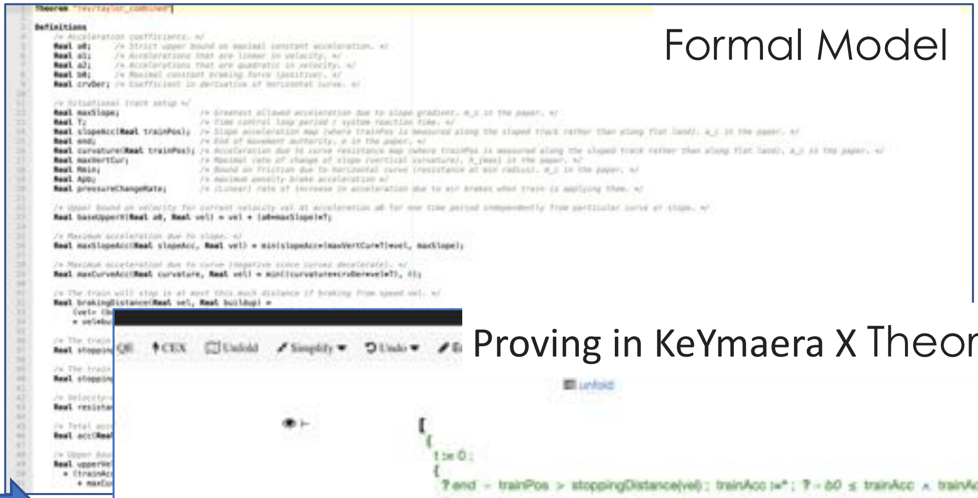
Formal Verification



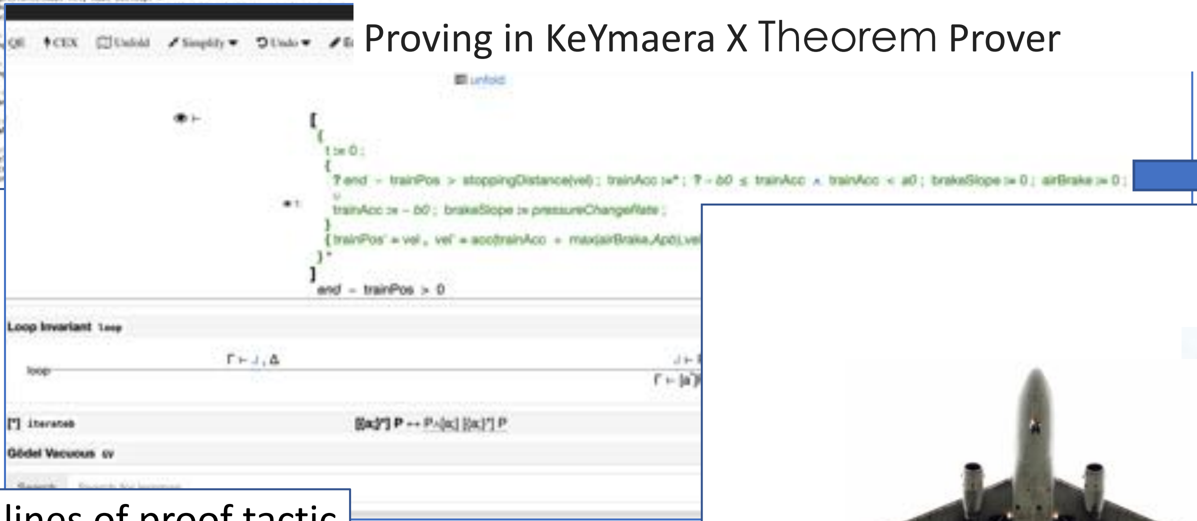
[1] J. Brosseau and B. M. Ede, “Development of an adaptive predictive braking enforcement algorithm”, Federal Railroad Administration, 2009.

Formal Verification

Complete FRA Model[1]

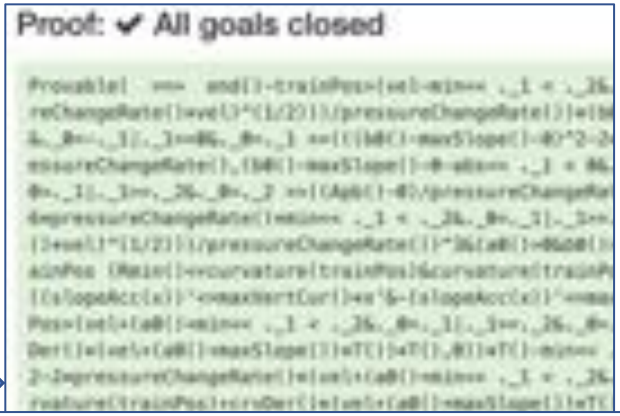


Formal Model



Proving in KeYmaera X Theorem Prover

2545 lines of proof tactic



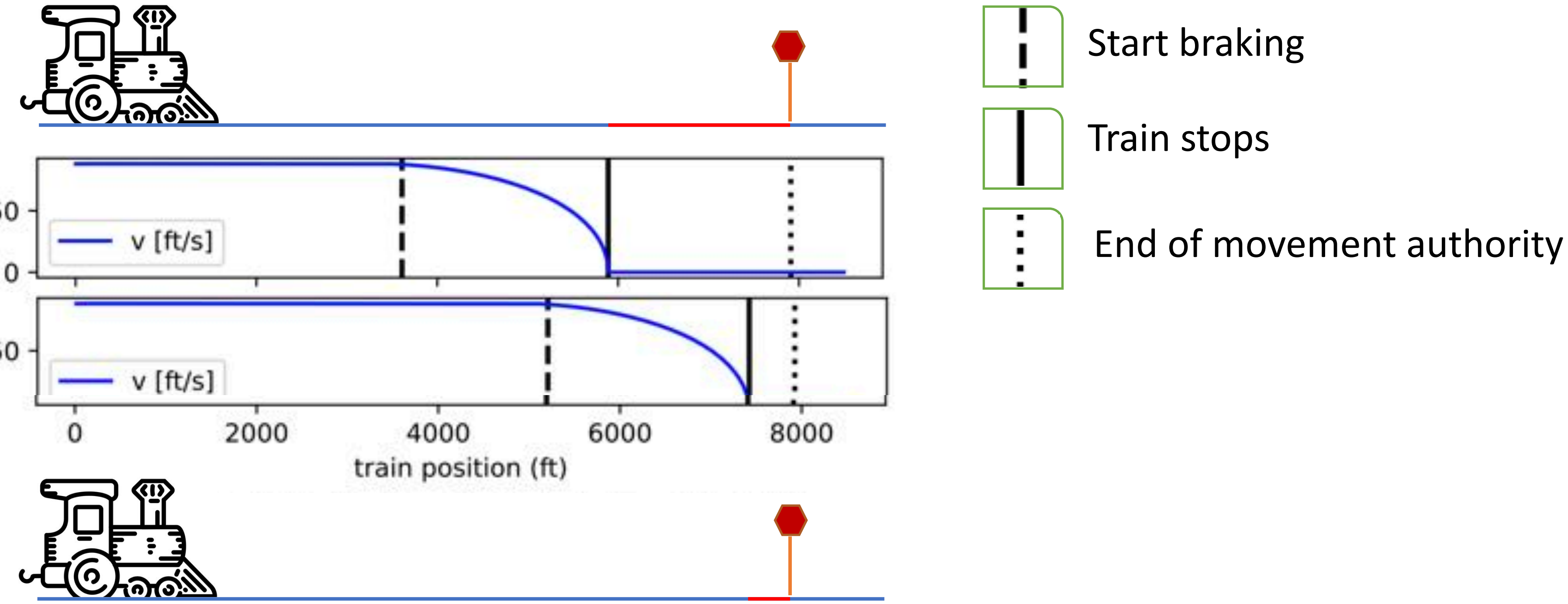
Generalizable

[1] J. Brosseau and B. M. Ede, “Development of an adaptive predicti
Administration, 2009.

Approach: Impact

Baseline[1]

Verified controller

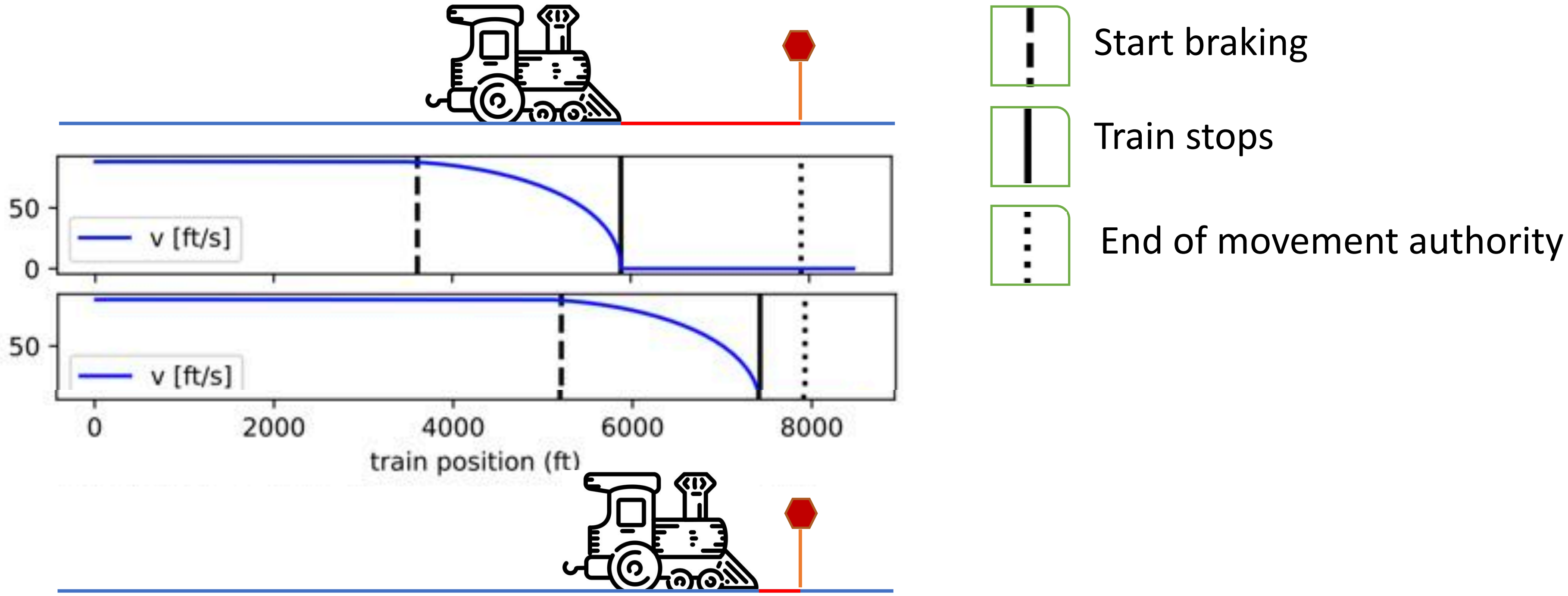


[1] J. Brosseau and B. M. Ede, “Development of an adaptive predictive braking enforcement algorithm”, Federal Railroad Administration, 2009.

Approach: Impact

Baseline[1]

Verified controller



[1] J. Brosseau and B. M. Ede, "Development of an adaptive predictive braking enforcement algorithm", Federal Railroad Administration, 2009.

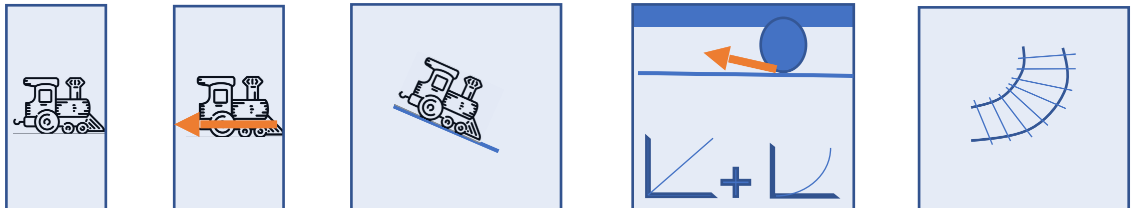
Overview

Part 1: Train Verification

- Introduction
- Techniques
- Controller
- Evaluation
- Summary

Background: Dynamics

Rate of change of train velocity is acceleration



$$p' = v, v' = a_l + a_a + a_s(p) + a_r(v) + a_c(p), a_b' = m_b$$

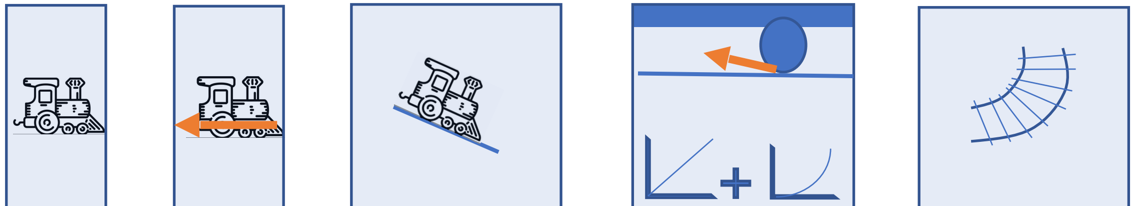
$$\text{with } a_l \in [-b_{\max}, a_{\max}], a_a = max(a_b, a_{b\max})$$

Rate of change of train position is velocity

Background: Dynamics

Rate of change of train velocity is acceleration

Air brakes ramp up



$$p' = v, v' = a_l + a_a + a_s(p) + a_r(v) + a_c(p), a_b' = m_b$$

$$\text{with } a_l \in [-b_{\max}, a_{\max}], a_a = max(a_b, a_{b\max})$$

Rate of change of train position is velocity

Unknown functions: slope, curve

$$p' = v, v' = a_l + a_a + a_s(p) + a_r(v) + a_c(p), a_b' = m_b$$

Unknown functions: slope, curve

Use worst case value …

m_s 0

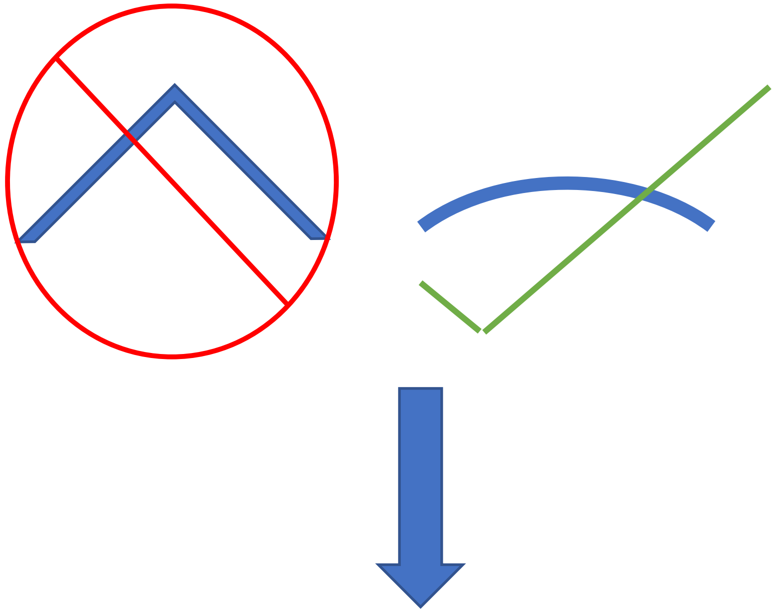
$$p' = v, v' = a_l + a_a + a_s(p) + a_r(v) + a_c(p), a'_b = m_b$$

Unknown function: replace with worst case value m_s

Unknown function: replace with worst case value 0

Unknown functions: slope, curve

… with improving estimates.

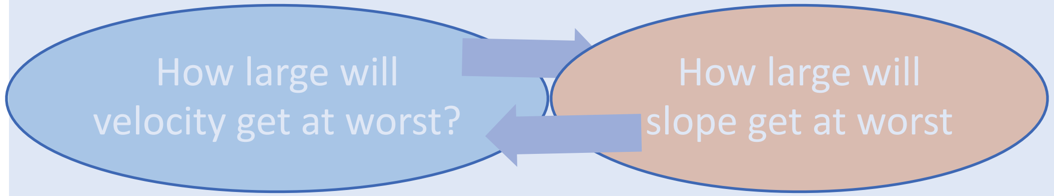


$$a_s(p) \leq \bar{a}_s(p_0) = \min(m_s, a_s(p_0) + u \cdot h_{\text{max}} \cdot T)$$

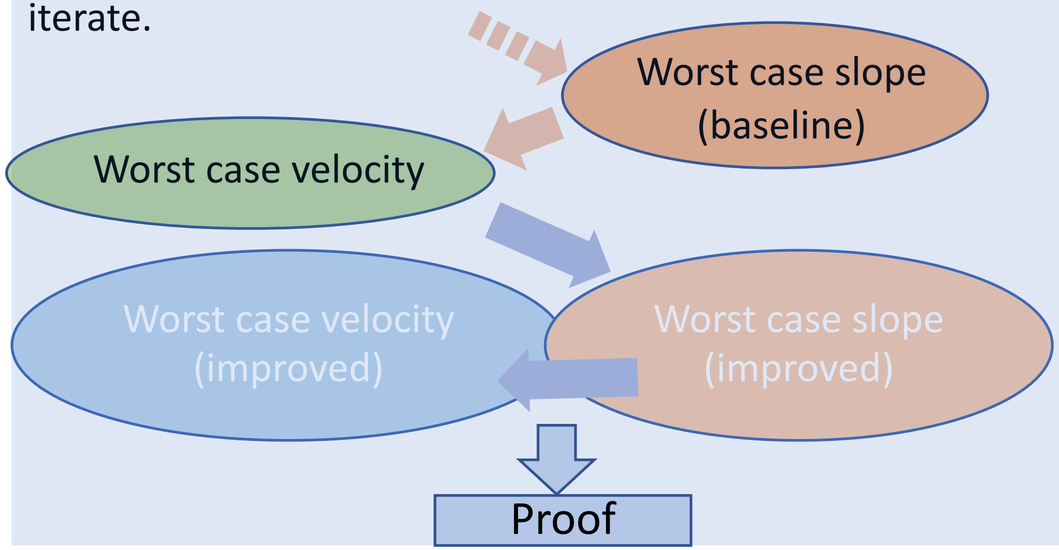
Other Proof Techniques

Circular Dependencies

Problem: Circular dependence while estimating worst case values.



Solution: Bootstrap cycle with naive values, then iterate.



Taylor Polynomial

Problem: Davis resistance integrates poorly.

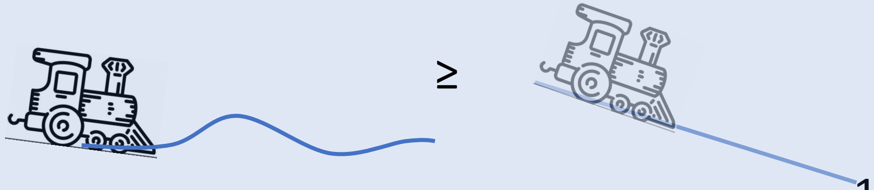
$$\frac{\left(\sqrt{4(a_l+m_s)a_2-a_1^2}\right)\cdot \tan\left(t\frac{\sqrt{4(a_l+m_s)a_2-a_1^2}}{2}+\tan^{-1}\left(\frac{a_1+2a_2v_0}{\sqrt{4(a_l+m_s)a_2-a_1^2}}\right)\right)-a_1}{2a_2}$$

Solution: Taylor polynomial approximation.

Ghost Trains

Problem: Intermediate reasoning steps transcendental.

Solution: Reason about as ODE (here represents dynamics of a "ghost" train).



Overview

- Introduction
- Techniques
- Controller
- Evaluation
- Summary

Control Structure

Control code runs in a loop with some latency T (in our case, to the order of a second).

End of *movement authority*: the train must stop by this point

$\{$

$t:=0;$

$\{$

$\{\ ?\ (\ e\text{-}p > \text{stoppingDistance}(p,\ v,\ a_b)\ );$

$a_l:=*;\ ?(-b_{max} \le a_l \le a_{max});$

Free Driving

$m_b:=0;\ a_b:=0;$

$\} \cup$

$\{\ a_l:=-b_{max};\ m_b:=m_p;\ \}$

Brake

$\}$

$\{\ p' = v,\ v' = a_l + \max(a_b,\ a_{bmax}) + a_s(p) + a_c(p) + a_r(v),\ a_b' = m_b,\ t' = 1$

$\&\ t \le T\ \&\ v \ge 0$

Train Motion for at most time T

$\}$

$\}^*$


```
Theorem "WP2/slopecurve_offset_airbrakes_1"

Definitions
  /* Acceleration coefficients. */
  Real a0;    /* Strict upper bound on maximal constant acceleration. */
  Real a1;    /* Accelerations that are linear in velocity. */
  Real a2;    /* Accelerations that are quadratic in velocity. */
  Real b0;    /* Maximal constant braking force (positive). */
  Real crvDer; /* Coefficient in derivative of horizontal curve. */

  /* Situational track setup */
  Real maxSlope;              /* Greatest allowed acceleration due to slope gradient. m_s in the paper.
  Real T;                     /* Time control loop period / system reaction time. */
  Real slopeAcc(Real trainPos);  /* Slope acceleration map (where trainPos is measured along the
sloped track rather than along flat land). a_c in the paper. */
  Real end;                   /* End of movement authority. e in the paper. */
  Real curvature(Real trainPos); /* Acceleration due to curve resistance map (where trainPos is
measured along the sloped track rather than along flat land). a_c in the paper. */
  Real maxVertCur;            /* Maximal rate of change of slope (vertical curvature). h_{max} in the
paper. */
  Real Rmin;                  /* Bound on friction due to horizontal curve (resistance at min radius). m_c
the paper. */
  Real Apb;                   /* maximum penalty brake acceleration */
  Real buildupThreshold;      /* Time offset until pressure brakes can be applied. */
  Real pressureChangeRate;    /* (Linear) rate of increase in acceleration due to air brakes when tr
is applying them. */

  /* Upper bound on velocity for current velocity vel at acceleration a0 for one time period
independently from particular curve or slope. */
  Real baseUpperV(Real a0, Real vel) = vel + (a0+maxSlope)*T;

  /* Maximum acceleration due to slope. */
  Real maxSlopeAcc(Real slopeAcc, Real vel) = min(slopeAcc+(maxVertCur*T)*vel, maxSlope);

  /* Maximum acceleration due to curve (negative since curves decelerate). */
  Real maxCurveAcc(Real curvature, Real vel) = min((curvature+crvDer*vel*T), 0);

  /* The train will stop in at most this much distance if braking from speed vel. */
  Real brakingDistance(Real vel, Real buildup) = min(vel^2/(2*(b0-maxSlope)),
    (vel-(buildupT(buildup)*(b0-maxSlope)))^2/(2*(b0-maxSlope)) + vel*buildupT(buildup) + 1/2*(b
maxSlope)*buildupT(buildup)^2);

  /* The train will stop in at most this much distance if it accelerates for a time period and then brak
*/
  Real stoppingDistance(Real trainPos, Real vel) = brakingDistance(upperVel(a0, vel,
slopeAcc(trainPos), curvature(trainPos)), 0) +  upperDist(vel, slopeAcc(trainPos), curvature(trainPos)

  /* Velocity-dependent part of the acceleration that is always negative (since resistance). a_r in the
paper. */
  Real resistance(Real vel) = a1*vel + a2*vel^2;

  /* Total acceleration acting on the train at any point for a controlled acceleration trainAcc at veloc
vel from position trainPos. */
  Real acc(Real trainAcc, Real vel, Real trainPos) = trainAcc + slopeAcc(trainPos) + resistance(vel) +
curvature(trainPos);

  /* Upper bound on new velocity (increased for one time period). */
  Real upperVel(Real trainAcc, Real vel, Real slopeAcc, Real curvature) = vel
   + (trainAcc+maxSlopeAcc(slopeAcc, baseUpperV(trainAcc, vel))
     + maxCurveAcc(curvature, baseUpperV(trainAcc, vel)))*T;

  /* Upper bound on distance covered for one time period under acceleration. */
  Real upperDist(Real vel, Real slopeAcc, Real curvature) = vel*T + 1/2*(
      a0 + maxSlopeAcc(slopeAcc, baseUpperV(a0, vel))
      + maxCurveAcc(curvature, baseUpperV(a0, vel)))*T^2;

  /* Time till pipe pressure is at maximum given current pressure. */
  Real buildupT(Real buildup) = (Apb-buildup)/pressureChangeRate;

  /* Utility functions for absolute values: True iff |x|<=y */
  Bool absLessEq(Real x, Real y) <-> (x<=y & -x<=y);

  /* Track is built correctly: all changes in slope along actual track are bounded by maximal vertical
curve and curve derivatives. */
  Bool limitedTrackChange() <-> \forall x' \forall x absLessEq((slopeAcc(x))', maxVertCur*x')
   & \forall x' \forall x absLessEq((curvature(x))', crvDer*x');

  /* Assumptions on constants. */
  Bool conditionsOnConsts() <->
      a0>0                          /* Strict upper bound on maximal constant acceleration, must b
positive. */
      & b0>0                        /* Maximal constant braking force (positive). */
      & a1<0                        /* Accelerations that are linear in velocity. */
      & a2<0                        /* Accelerations that are quadratic in velocity. */
      & maxSlope>=0                 /* Greatest allowed acceleration due to slope gradient. */
      & T>0                         /* Time control loop period / system reaction time. */
      & b0-maxSlope>0               /* Brakes more powerful than effect of slope. */
      & a0-maxSlope>0               /* Engine more powerful than effect of slope. */
      & a0-maxSlope+Rmin>0          /* Engine more powerful than curve and slope. */
      & maxVertCur>=0               /* Maximal rate of change of slope (vertical curvature). *
      & Rmin<=0                     /* Bound on friction due to horizontal curve (resistance at m
radius). */
      & crvDer>=0                   /* Coefficient in derivative of horizontal curve. */
      & \forall trainPos (Rmin<=curvature(trainPos) & curvature(trainPos)<=0)  /* Track is built correc
all curvatures along actual track satisfy bound on friction due to curve. */
      & \forall trainPos absLessEq(slopeAcc(trainPos), maxSlope)      /* Track is built correctly: all slop
accelerations along actual track are within maxSlope. */
      & Apb<0                       /* Maximally engaged air pressure brakes provide negative
acceleration. */
      & buildupThreshold>0          /* Air pressure brake propagation time is some positive
numbers. */
      & pressureChangeRate<0;       /* The rate at which acceleration provided by the
pressure brakes increases. */
End.

ProgramVariables
   Real trainPos;    /* The position of the train. */
   Real vel;         /* The speed of the train. */
   Real trainAcc;    /* Acceleration/deceleration control, i.e., engine acceleration and braking. */
   Real buildup;     /* How long pipe pressure has been building up. */
   Real brakeSlope;  /* pressureChangeRate if controller has chosen to brake, 0 otherwise. */
   Real airBrake;    /* Acceleration due to current pipe pressure. */
End.

Problem
 end-trainPos>brakingDistance(vel, 0)
 & conditionsOnConsts()
 & limitedTrackChange()
 & vel>=0
 & brakeSlope=0
 & airBrake=0
 ->
 [
     {
       t:=0;
       {{?(end-trainPos > stoppingDistance(trainPos, vel));
          trainAcc:=*;
```

Control Structure

Control code runs in a loop with some latency T (in our case, to the order of a second).

{

$t:=0;$

{

{ ? ($e - p >$ stoppingDistance(p, v, a_b));

$a_l:=*;\ ?(-b_{max} \leq a_l \leq a_{max});$
$m_b:=0;\ a_b:=0;$

Free Driving

} ∪

{ $a_l:=-b_{max};\ m_b:=m_p;$ }

Brake

}

{ $p' = v,\ v' = a_l + \max\left(a_b,\ a_{bmax}\right) + a_s(p) + a_c(p) + a_r(v),\ a_b' = m_b,\ t' = 1$
$\&\ t \leq T\ \&\ v \geq 0$

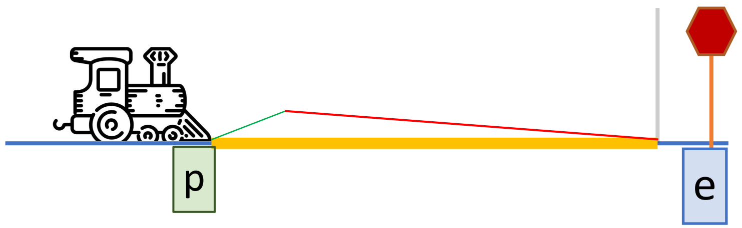
Train Motion for at most time T

}

}*

End of *movement authority*: the train must stop by this point

Upper bound on distance traveled before train stops if you accelerate for one control cycle and brake after that.




```
Theorem "WP2/slopecurve_offset_airbrakes_1"

Definitions
  /* Acceleration coefficients. */
  Real a0;    /* Strict upper bound on maximal constant acceleration. */
  Real a1;    /* Accelerations that are linear in velocity. */
  Real a2;    /* Accelerations that are quadratic in velocity. */
  Real b0;    /* Maximal constant braking force (positive). */
  Real crvDer; /* Coefficient in derivative of horizontal curve. */

  /* Situational track setup */
  Real maxSlope;             /* Greatest allowed acceleration due to slope gradient. m_s in the paper.
  Real T;                    /* Time control loop period / system reaction time. */
  Real slopeAcc(Real trainPos);  /* Slope acceleration map (where trainPos is measured along the
sloped track rather than along flat land). a_c in the paper. */
  Real end;                  /* End of movement authority. e in the paper. */
  Real curvature(Real trainPos); /* Acceleration due to curve resistance map (where trainPos is
measured along the sloped track rather than along flat land). a_c in the paper. */
  Real maxVertCur;           /* Maximal rate of change of slope (vertical curvature). h_{max} in the
paper. */
  Real Rmin;                 /* Bound on friction due to horizontal curve (resistance at min radius). m_c
the paper. */
  Real Apb;                  /* maximum penalty brake acceleration */
  Real buildupThreshold;     /* Time offset until pressure brakes can be applied. */
  Real pressureChangeRate;   /* (Linear) rate of increase in acceleration due to air brakes when tr
is applying them. */

  /* Upper bound on velocity for current velocity vel at acceleration a0 for one time period
independently from particular curve or slope. */
  Real baseUpperV(Real a0, Real vel) = vel + (a0+maxSlope)*T;

  /* Maximum acceleration due to slope. */
  Real maxSlopeAcc(Real slopeAcc, Real vel) = min(slopeAcc+(maxVertCur*T)*vel, maxSlope);

  /* Maximum acceleration due to curve (negative since curves decelerate). */
  Real maxCurveAcc(Real curvature, Real vel) = min((curvature+crvDer*vel*T), 0);

  /* The train will stop in at most this much distance if braking from speed vel. */
  Real brakingDistance(Real vel, Real buildup) = min(vel^2/(2*(b0-maxSlope)),
    (vel-(buildupT(buildup)*(b0-maxSlope)))^2/(2*(b0-maxSlope)) + vel*buildupT(buildup) + 1/2*(b
maxSlope)*buildupT(buildup)^2);

  /* The train will stop in at most this much distance if it accelerates for a time period and then brak
*/
  Real stoppingDistance(Real trainPos, Real vel) = brakingDistance(upperVel(a0, vel,
slopeAcc(trainPos), curvature(trainPos)), 0) +  upperDist(vel, slopeAcc(trainPos), curvature(trainPos)

  /* Velocity-dependent part of the acceleration that is always negative (since resistance). a_r in the
paper. */
  Real resistance(Real vel) = a1*vel + a2*vel^2;

  /* Total acceleration acting on the train at any point for a controlled acceleration trainAcc at veloc
vel from position trainPos. */
  Real acc(Real trainAcc, Real vel, Real trainPos) = trainAcc + slopeAcc(trainPos) + resistance(vel) +
curvature(trainPos);

  /* Upper bound on new velocity (increased for one time period). */
  Real upperVel(Real trainAcc, Real vel, Real slopeAcc, Real curvature) = vel
   + (trainAcc+maxSlopeAcc(slopeAcc, baseUpperV(trainAcc, vel))
    + maxCurveAcc(curvature, baseUpperV(trainAcc, vel)))*T;

  /* Upper bound on distance covered for one time period under acceleration. */
  Real upperDist(Real vel, Real slopeAcc, Real curvature) = vel*T + 1/2*(
     a0 + maxSlopeAcc(slopeAcc, baseUpperV(a0, vel))
     + maxCurveAcc(curvature, baseUpperV(a0, vel)))*T^2;

  /* Time till pipe pressure is at maximum given current pressure. */
  Real buildupT(Real buildup) = (Apb-buildup)/pressureChangeRate;

  /* Utility functions for absolute values: True iff |x|<=y */
  Bool absLessEq(Real x, Real y) <-> (x<=y & -x<=y);

  /* Track is built correctly: all changes in slope along actual track are bounded by maximal vertical
curve and curve derivatives. */
  Bool limitedTrackChange() <-> \forall x' \forall x absLessEq((slopeAcc(x))', maxVertCur*x')
   & \forall x' \forall x absLessEq((curvature(x))', crvDer*x');

  /* Assumptions on constants. */
  Bool conditionsOnConsts() <->
     a0>0                        /* Strict upper bound on maximal constant acceleration, must b
positive. */
     & b0>0                      /* Maximal constant braking force (positive). */
     & a1<0                      /* Accelerations that are linear in velocity. */
     & a2<0                      /* Accelerations that are quadratic in velocity. */
     & maxSlope>=0                  /* Greatest allowed acceleration due to slope gradient. */
     & T>0                       /* Time control loop period / system reaction time. */
     & b0-maxSlope>0                /* Brakes more powerful than effect of slope. */
     & a0-maxSlope>0                /* Engine more powerful than effect of slope. */
     & a0-maxSlope+Rmin>0              /* Engine more powerful than curve and slope. */
     & maxVertCur>=0                /* Maximal rate of change of slope (vertical curvature). *
     & Rmin<=0                    /* Bound on friction due to horizontal curve (resistance at m
radius). */
     & crvDer>=0                  /* Coefficient in derivative of horizontal curve. */
     & \forall trainPos (Rmin<=curvature(trainPos) & curvature(trainPos)<=0)  /* Track is built correc
all curvatures along actual track satisfy bound on friction due to curve. */
     & \forall trainPos absLessEq(slopeAcc(trainPos), maxSlope)     /* Track is built correctly: all slop
accelerations along actual track are within maxSlope. */
     & Apb<0                      /* Maximally engaged air pressure brakes provide negative
acceleration. */
     & buildupThreshold>0              /* Air pressure brake propagation time is some positive
numbers. */
     & pressureChangeRate<0;             /* The rate at which acceleration provided by the
pressure brakes increases. */
End.

ProgramVariables
  Real trainPos;    /* The position of the train. */
  Real vel;       /* The speed of the train. */
  Real trainAcc;    /* Acceleration/deceleration control, i.e., engine acceleration and braking. */
  Real buildup;     /* How long pipe pressure has been building up. */
  Real brakeSlope;   /* pressureChangeRate if controller has chosen to brake, 0 otherwise. */
  Real airBrake;    /* Acceleration due to current pipe pressure. */
End.

Problem
 end-trainPos>brakingDistance(vel, 0)
 & conditionsOnConsts()
 & limitedTrackChange()
 & vel>=0
 & brakeSlope=0
 & airBrake=0
 ->
 [
   {
     t:=0;
     {{?(end-trainPos > stoppingDistance(trainPos, vel));
       trainAcc:=*;
```

Control Structure

Control code runs in a loop with some latency T (in our case, to the order of a second).

{

  $t := 0;$

  {

    $\{ ?(e\text{-}p > \text{stoppingDistance}(p, v, a_b));$ — Only if there is a sufficient distance margin

    [redacted] Free Driving — Allow acceleration

    $\} \cup$

    [redacted] Brake — Always allow braking

  }

  [redacted] Train Motion for at most time T

  }

}*

```
Theorem "WP2/slopecurve_offset_airbrakes_1"

Definitions
  /* Acceleration coefficients. */
  Real a0;    /* Strict upper bound on maximal constant acceleration. */
  Real a1;    /* Accelerations that are linear in velocity. */
  Real a2;    /* Accelerations that are quadratic in velocity. */
  Real b0;    /* Maximal constant braking force (positive). */
  Real crvDer; /* Coefficient in derivative of horizontal curve. */

  /* Situational track setup */
  Real maxSlope;            /* Greatest allowed acceleration due to slope gradient. m_s in the paper.
  Real T;                   /* Time control loop period / system reaction time. */
  Real slopeAcc(Real trainPos);  /* Slope acceleration map (where trainPos is measured along the
sloped track rather than along flat land). a_c in the paper. */
  Real end;                 /* End of movement authority. e in the paper. */
  Real curvature(Real trainPos); /* Acceleration due to curve resistance map (where trainPos is
measured along the sloped track rather than along flat land). a_c in the paper. */
  Real maxVertCur;          /* Maximal rate of change of slope (vertical curvature). h_{max} in the
paper. */
  Real Rmin;                /* Bound on friction due to horizontal curve (resistance at min radius). m_c
the paper. */
  Real Apb;                 /* maximum penalty brake acceleration */
  Real buildupThreshold;       /* Time offset until pressure brakes can be applied. */
  Real pressureChangeRate;      /* (Linear) rate of increase in acceleration due to air brakes when tr
is applying them. */

  /* Upper bound on velocity for current velocity vel at acceleration a0 for one time period
independently from particular curve or slope. */
  Real baseUpperV(Real a0, Real vel) = vel + (a0+maxSlope)*T;

  /* Maximum acceleration due to slope. */
  Real maxSlopeAcc(Real slopeAcc, Real vel) = min(slopeAcc+(maxVertCur*T)*vel, maxSlope);

  /* Maximum acceleration due to curve (negative since curves decelerate). */
  Real maxCurveAcc(Real curvature, Real vel) = min((curvature+crvDer*vel*T), 0);

  /* The train will stop in at most this much distance if braking from speed vel. */
  Real brakingDistance(Real vel, Real buildup) = min(vel^2/(2*(b0-maxSlope)),
    (vel-(buildupT(buildup)*(b0-maxSlope)))^2/(2*(b0-maxSlope)) + vel*buildupT(buildup) + 1/2*(b
maxSlope)*buildupT(buildup)^2);

  /* The train will stop in at most this much distance if it accelerates for a time period and then brak
*/
  Real stoppingDistance(Real trainPos, Real vel) = brakingDistance(upperVel(a0, vel,
slopeAcc(trainPos), curvature(trainPos)), 0) +  upperDist(vel, slopeAcc(trainPos), curvature(trainPos)

  /* Velocity-dependent part of the acceleration that is always negative (since resistance). a_r in the
paper. */
  Real resistance(Real vel) = a1*vel + a2*vel^2;

  /* Total acceleration acting on the train at any point for a controlled acceleration trainAcc at veloc
vel from position trainPos. */
  Real acc(Real trainAcc, Real vel, Real trainPos) = trainAcc + slopeAcc(trainPos) + resistance(vel) +
curvature(trainPos);

  /* Upper bound on new velocity (increased for one time period). */
  Real upperVel(Real trainAcc, Real vel, Real slopeAcc, Real curvature) = vel
   + (trainAcc+maxSlopeAcc(slopeAcc, baseUpperV(trainAcc, vel))
     + maxCurveAcc(curvature, baseUpperV(trainAcc, vel)))*T;

  /* Upper bound on distance covered for one time period under acceleration. */
  Real upperDist(Real vel, Real slopeAcc, Real curvature) = vel*T + 1/2*(
      a0 + maxSlopeAcc(slopeAcc, baseUpperV(a0, vel))
      + maxCurveAcc(curvature, baseUpperV(a0, vel)))*T^2;

  /* Time till pipe pressure is at maximum given current pressure. */
  Real buildupT(Real buildup) = (Apb-buildup)/pressureChangeRate;

  /* Utility functions for absolute values: True iff |x|<=y */
  Bool absLessEq(Real x, Real y) <-> (x<=y & -x<=y);

  /* Track is built correctly: all changes in slope along actual track are bounded by maximal vertical
curve and curve derivatives. */
  Bool limitedTrackChange() <-> \forall x' \forall x absLessEq((slopeAcc(x))', maxVertCur*x')
   & \forall x' \forall x absLessEq((curvature(x))', crvDer*x');

  /* Assumptions on constants. */
  Bool conditionsOnConsts() <->
     a0>0                          /* Strict upper bound on maximal constant acceleration, must b
positive. */
     & b0>0                        /* Maximal constant braking force (positive). */
     & a1<0                        /* Accelerations that are linear in velocity. */
     & a2<0                        /* Accelerations that are quadratic in velocity. */
     & maxSlope>=0                      /* Greatest allowed acceleration due to slope gradient. */
     & T>0                         /* Time control loop period / system reaction time. */
     & b0-maxSlope>0                    /* Brakes more powerful than effect of slope. */
     & a0-maxSlope>0                    /* Engine more powerful than effect of slope. */
     & a0-maxSlope+Rmin>0                 /* Engine more powerful than curve and slope. */
     & maxVertCur>=0                    /* Maximal rate of change of slope (vertical curvature). *
     & Rmin<=0                       /* Bound on friction due to horizontal curve (resistance at m
radius). */
     & crvDer>=0                     /* Coefficient in derivative of horizontal curve. */
     & \forall trainPos (Rmin<=curvature(trainPos) & curvature(trainPos)<=0)  /* Track is built correc
all curvatures along actual track satisfy bound on friction due to curve. */
     & \forall trainPos absLessEq(slopeAcc(trainPos), maxSlope)      /* Track is built correctly: all slop
accelerations along actual track are within maxSlope. */
     & Apb<0                       /* Maximally engaged air pressure brakes provide negative
acceleration. */
     & buildupThreshold>0                  /* Air pressure brake propagation time is some positive
numbers. */
     & pressureChangeRate<0;                 /* The rate at which acceleration provided by the
pressure brakes increases. */
End.

ProgramVariables
  Real trainPos;    /* The position of the train. */
  Real vel;       /* The speed of the train. */
  Real trainAcc;    /* Acceleration/deceleration control, i.e., engine acceleration and braking. */
  Real buildup;     /* How long pipe pressure has been building up. */
  Real brakeSlope;   /* pressureChangeRate if controller has chosen to brake, 0 otherwise. */
  Real airBrake;    /* Acceleration due to current pipe pressure. */
End.

Problem
 end-trainPos>brakingDistance(vel, 0)
 & conditionsOnConsts()
 & limitedTrackChange()
 & vel>=0
 & brakeSlope=0
 & airBrake=0
 ->
 [
    {
      t:=0;
      {{?(end-trainPos > stoppingDistance(trainPos, vel));
        trainAcc:=*;
```

Control Structure

Control code runs in a loop with some latency T (in our case, to the order of a second).

{

t:=0;

{

{ ? (e-p > stoppingDistance(p, v, a_b));

a_l:=*; ?($-b_{max} \le a_l \le a_{max}$);

Free Driving

m_b:=0; a_b:=0;

} ∪

{ a_l:=$-b_{max}$; m_b Brake $_p$; }

Control Envelope

}

{ $p' = v,\ v' = a_l + \max\left(a_b,\ a_{bmax}\right) + a_s\left(p\right) + a_c\left(p\right) + a_r(v),\ a_b' = m_b,\ t' = 1$

Train Motion for at most time T

& $t \le T$ & $v \ge 0$

}

}*

```
Theorem "WP2/slopecurve_offset_airbrakes_1"

Definitions
  /* Acceleration coefficients. */
  Real a0;    /* Strict upper bound on maximal constant acceleration. */
  Real a1;    /* Accelerations that are linear in velocity. */
  Real a2;    /* Accelerations that are quadratic in velocity. */
  Real b0;    /* Maximal constant braking force (positive). */
  Real crvDer; /* Coefficient in derivative of horizontal curve. */

  /* Situational track setup */
  Real maxSlope;              /* Greatest allowed acceleration due to slope gradient. m_s in the paper.
  Real T;                     /* Time control loop period / system reaction time. */
  Real slopeAcc(Real trainPos);  /* Slope acceleration map (where trainPos is measured along the
sloped track rather than along flat land). a_c in the paper. */
  Real end;                   /* End of movement authority. e in the paper. */
  Real curvature(Real trainPos); /* Acceleration due to curve resistance map (where trainPos is
measured along the sloped track rather than along flat land). a_c in the paper. */
  Real maxVertCur;            /* Maximal rate of change of slope (vertical curvature). h_{max} in the
paper. */
  Real Rmin;                  /* Bound on friction due to horizontal curve (resistance at min radius). m_c
the paper. */
  Real Apb;                   /* maximum penalty brake acceleration */
  Real buildupThreshold;      /* Time offset until pressure brakes can be applied. */
  Real pressureChangeRate;    /* (Linear) rate of increase in acceleration due to air brakes when tr
is applying them. */

  /* Upper bound on velocity for current velocity vel at acceleration a0 for one time period
independently from particular curve or slope. */
  Real baseUpperV(Real a0, Real vel) = vel + (a0+maxSlope)*T;

  /* Maximum acceleration due to slope. */
  Real maxSlopeAcc(Real slopeAcc, Real vel) = min(slopeAcc+(maxVertCur*T)*vel, maxSlope);

  /* Maximum acceleration due to curve (negative since curves decelerate). */
  Real maxCurveAcc(Real curvature, Real vel) = min((curvature+crvDer*vel*T), 0);

  /* The train will stop in at most this much distance if braking from speed vel. */
  Real brakingDistance(Real vel, Real buildup) = min(vel^2/(2*(b0-maxSlope)),
    (vel-(buildupT(buildup)*(b0-maxSlope)))^2/(2*(b0-maxSlope)) + vel*buildupT(buildup) + 1/2*(b
maxSlope)*buildupT(buildup)^2);

  /* The train will stop in at most this much distance if it accelerates for a time period and then brak
*/
  Real stoppingDistance(Real trainPos, Real vel) = brakingDistance(upperVel(a0, vel,
slopeAcc(trainPos), curvature(trainPos)), 0) +  upperDist(vel, slopeAcc(trainPos), curvature(trainPos)

  /* Velocity-dependent part of the acceleration that is always negative (since resistance). a_r in the
paper. */
  Real resistance(Real vel) = a1*vel + a2*vel^2;

  /* Total acceleration acting on the train at any point for a controlled acceleration trainAcc at veloc
vel from position trainPos. */
  Real acc(Real trainAcc, Real vel, Real trainPos) = trainAcc + slopeAcc(trainPos) + resistance(vel) +
curvature(trainPos);

  /* Upper bound on new velocity (increased for one time period). */
  Real upperVel(Real trainAcc, Real vel, Real slopeAcc, Real curvature) = vel
   + (trainAcc+maxSlopeAcc(slopeAcc, baseUpperV(trainAcc, vel))
     + maxCurveAcc(curvature, baseUpperV(trainAcc, vel)))*T;

  /* Upper bound on distance covered for one time period under acceleration. */
  Real upperDist(Real vel, Real slopeAcc, Real curvature) = vel*T + 1/2*(
      a0 + maxSlopeAcc(slopeAcc, baseUpperV(a0, vel))
      + maxCurveAcc(curvature, baseUpperV(a0, vel)))*T^2;

  /* Time till pipe pressure is at maximum given current pressure. */
  Real buildupT(Real buildup) = (Apb-buildup)/pressureChangeRate;

  /* Utility functions for absolute values: True iff |x|<=y */
  Bool absLessEq(Real x, Real y) <-> (x<=y & -x<=y);

  /* Track is built correctly: all changes in slope along actual track are bounded by maximal vertical
curve and curve derivatives. */
  Bool limitedTrackChange() <-> \forall x' \forall x absLessEq((slopeAcc(x))', maxVertCur*x')
   & \forall x' \forall x absLessEq((curvature(x))', crvDer*x');

  /* Assumptions on constants. */
  Bool conditionsOnConsts() <->
     a0>0                        /* Strict upper bound on maximal constant acceleration, must b
positive. */
     & b0>0                      /* Maximal constant braking force (positive). */
     & a1<0                      /* Accelerations that are linear in velocity. */
     & a2<0                      /* Accelerations that are quadratic in velocity. */
     & maxSlope>=0               /* Greatest allowed acceleration due to slope gradient. */
     & T>0                       /* Time control loop period / system reaction time. */
     & b0-maxSlope>0             /* Brakes more powerful than effect of slope. */
     & a0-maxSlope>0             /* Engine more powerful than effect of slope. */
     & a0-maxSlope+Rmin>0        /* Engine more powerful than curve and slope. */
     & maxVertCur>=0             /* Maximal rate of change of slope (vertical curvature). *
     & Rmin<=0                   /* Bound on friction due to horizontal curve (resistance at m
radius). */
     & crvDer>=0                 /* Coefficient in derivative of horizontal curve. */
     & \forall trainPos (Rmin<=curvature(trainPos) & curvature(trainPos)<=0)  /* Track is built correc
all curvatures along actual track satisfy bound on friction due to curve. */
     & \forall trainPos absLessEq(slopeAcc(trainPos), maxSlope)      /* Track is built correctly: all slop
accelerations along actual track are within maxSlope. */
     & Apb<0                     /* Maximally engaged air pressure brakes provide negative
acceleration. */
     & buildupThreshold>0        /* Air pressure brake propagation time is some positive
numbers. */
     & pressureChangeRate<0;     /* The rate at which acceleration provided by the
pressure brakes increases. */
End.

ProgramVariables
  Real trainPos;    /* The position of the train. */
  Real vel;         /* The speed of the train. */
  Real trainAcc;    /* Acceleration/deceleration control, i.e., engine acceleration and braking. */
  Real buildup;     /* How long pipe pressure has been building up. */
  Real brakeSlope;  /* pressureChangeRate if controller has chosen to brake, 0 otherwise. */
  Real airBrake;    /* Acceleration due to current pipe pressure. */
End.

Problem
 end-trainPos>brakingDistance(vel, 0)
 & conditionsOnConsts()
 & limitedTrackChange()
 & vel>=0
 & brakeSlope=0
 & airBrake=0
 ->
 [
   {
     t:=0;
     {{?(end-trainPos > stoppingDistance(trainPos, vel));
       trainAcc:=*;
```

Envelope: Where the Complexity is

$$\texttt{brakeDist}_a(v,a_b) =$$

$$vt_b(v,a_b) + \frac{1}{2}(b_{\max} - m_s + a_b)t_b(v,a_b)^2 + \frac{1}{6}(m_p)t_b(v,a_b)^3$$

$$+ \frac{v - (b_{\max} - m_s + a_b)t_b(v,a_b) + \frac{1}{2}m_p t_b(v,a_b)^2}{2(b_{\max} - m_s - a_{b\max})}$$

$$t_b(v,a_b) = \min((a_{b\max} - a_b)/m_p,$$

$$\left.\frac{(b_{\max} - m_s + a_b) - |(b_{\max} - m_s + a_b)^2 - 2m_p v|}{m_p}\right)$$

$$\texttt{stopDist}_a(p,v,a_b) = vT + \left(\frac{a_{\max} + \bar{a}_s(p)}{2} + \frac{\bar{a}_c(p)}{2}\right)T^2$$

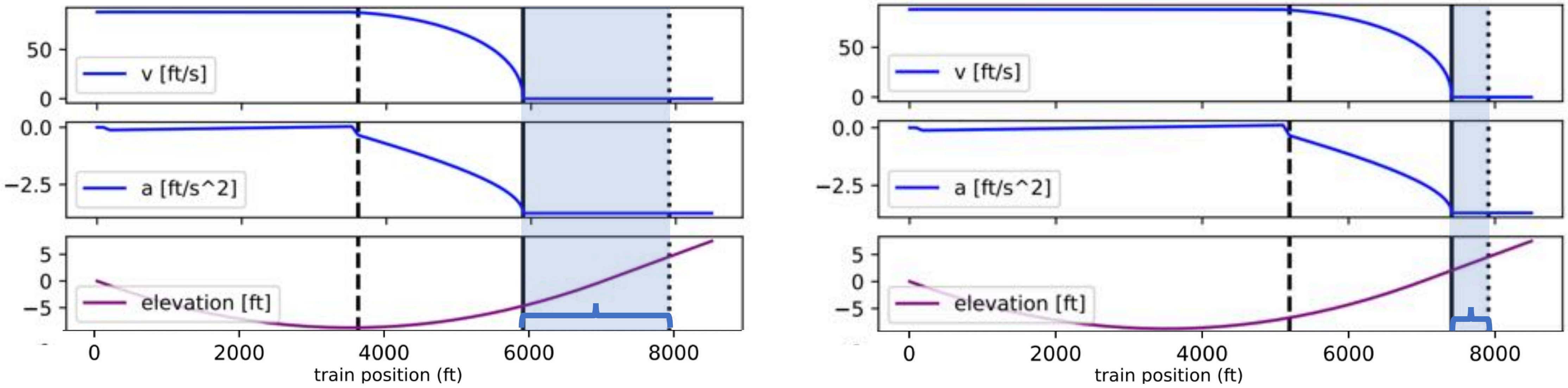
$$+ \texttt{brakeDist}_a\left(\left(v + (a_{\max} + \bar{a}_s(p) + \bar{a}_c(p))T\right)^2, 0\right)$$

Overview

Part 1: Train Verification

- Introduction
- Techniques
- Controller
- **Evaluation**
- Summary

Limiting Undershoot while Maintaining Safety



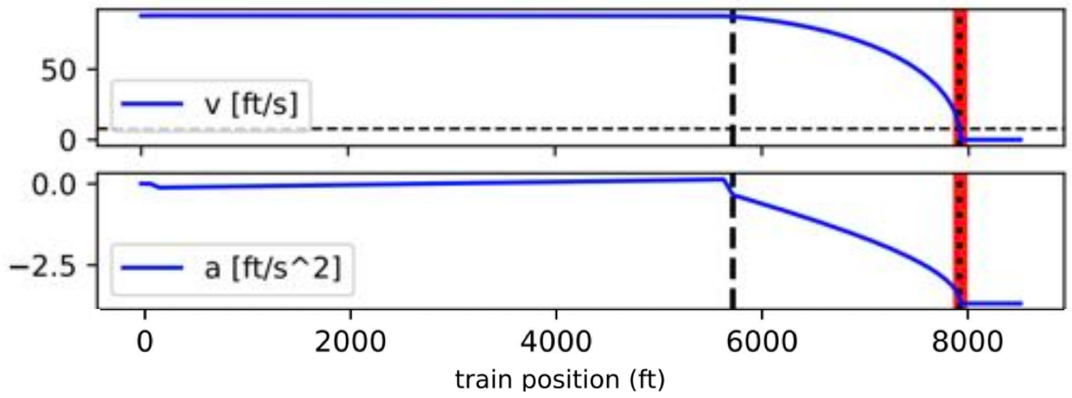
Start braking

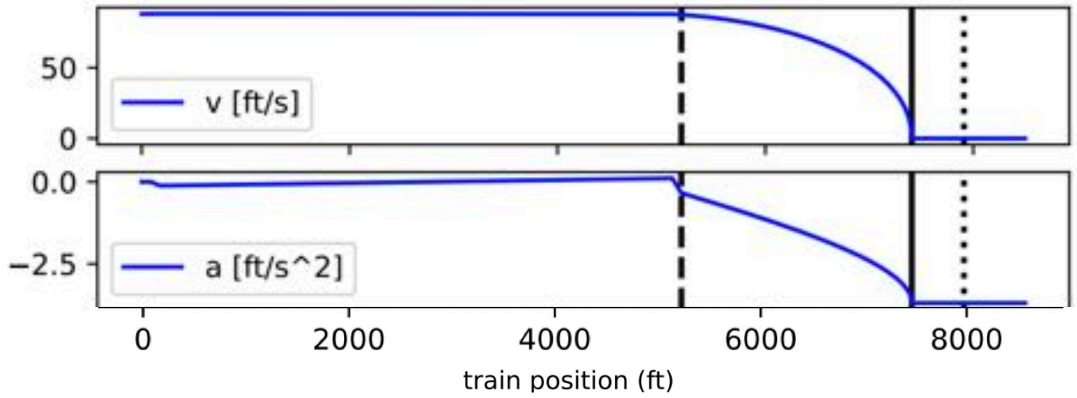
End of movement authority

Train stops

[1] J. Brosseau and B. M. Ede, "Development of an adaptive predictive braking enforcement algorithm", Federal Railroad Administration, 2009.

Limiting Undershoot while Maintaining Safety





Start braking

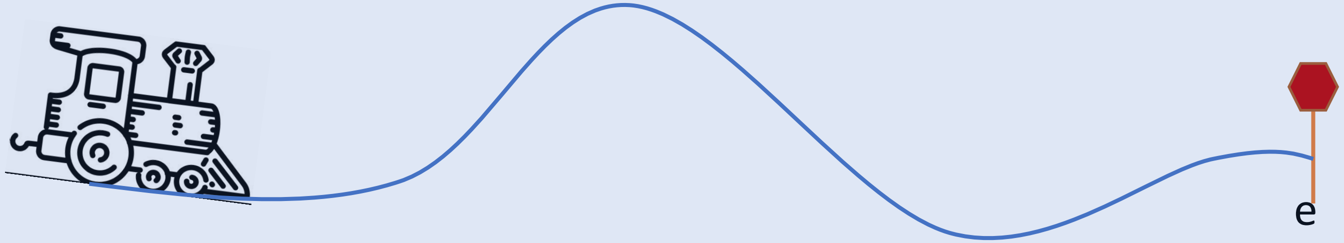
End of movement authority

Train stops

[1] J. Brosseau and B. M. Ede, “Development of an adaptive predictive braking enforcement algorithm”, Federal Railroad Administration, 2009.

Summary

Proofs: https://doi.org/10.1184/R1/19542610



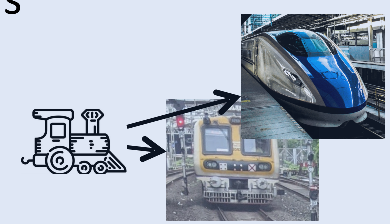
Verified controller for full FRA model dynamics. KeYmaera X proofs available online

Generalizable Techniques

- Dealing with unknown functions
- Circular dependencies
- Taylor polynomials
- Ghost dynamics

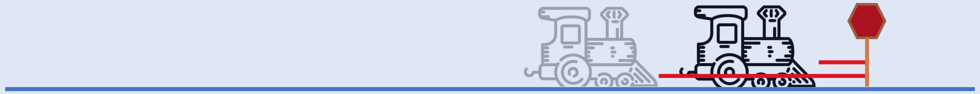
Verified Model Generalizability

- Abstraction of physical details
- Nondeterministic controller



Experiments

Controller limits undershoot while maintaining safety



Pt 2: CESAR: Control Envelope Synthesis via Angelic Refinements

Aditi Kabra Jonathan Laurent Stefan Mitsch André Platzer

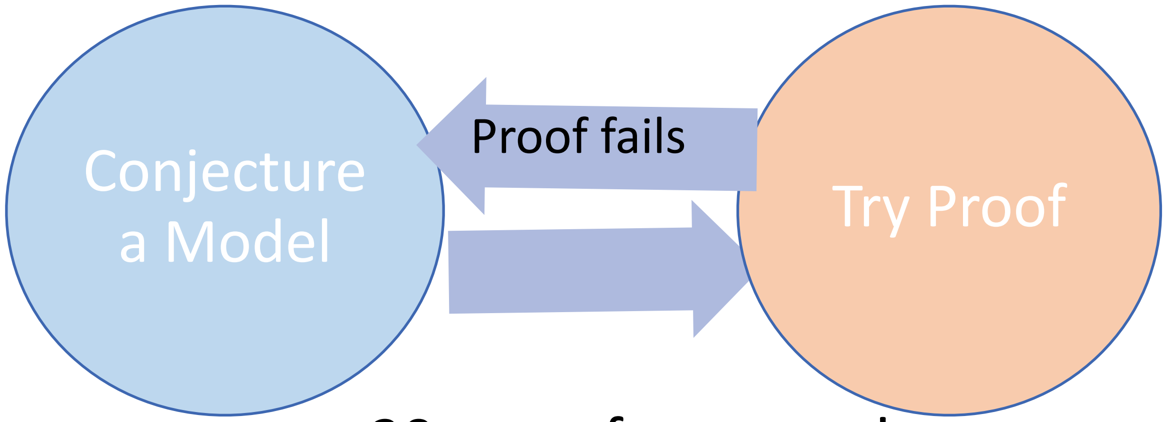
Overview

Part 2: Synthesis

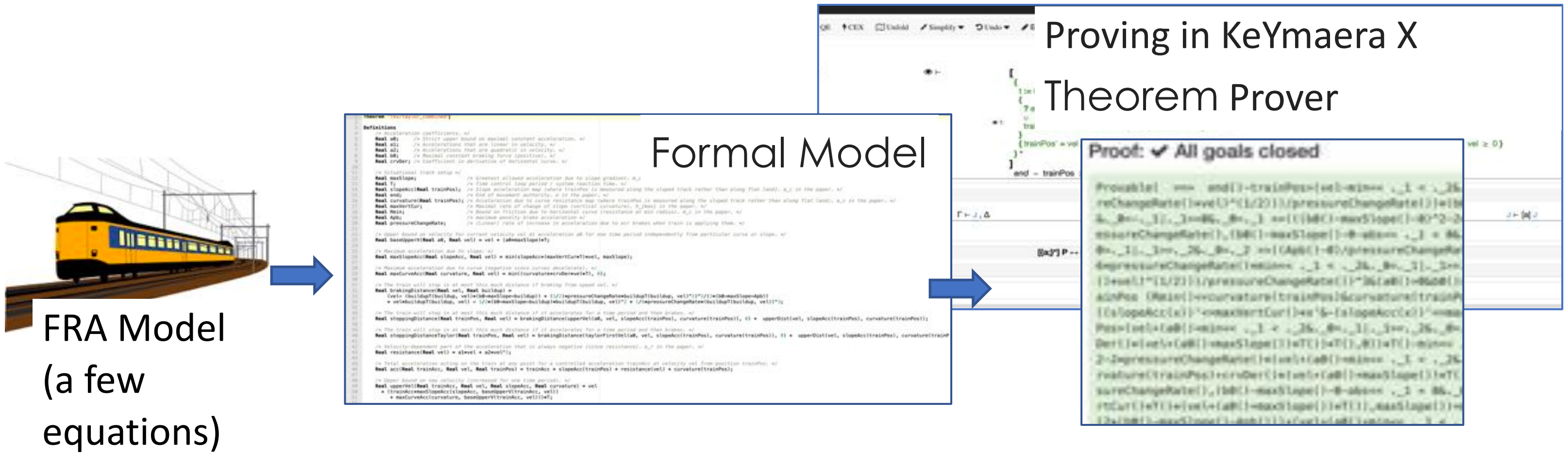
- **Introduction**
- Problem Statement
- Game Logic and Solution
- Refinement
- Evaluation

Design by proof

Can we automate it?

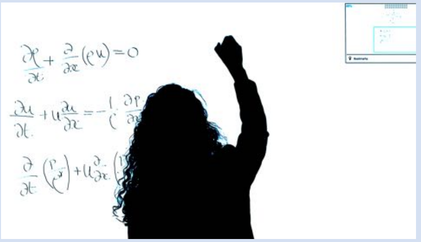


30+ proof attempts!



Synthesis Pipeline

Model Template



Synthesis procedure fills out the hard parts

Control Envelope

Related work

Other Work	This Work
Controller Synthesis Techniques 7. Belta, C., Yordanov, B., Gol, E.A.: Formal Methods for Discrete-Time Dynamical Systems. Springer Cham (2017) 21. Liu, S., Trivedi, A., Yin, X., Zamani, M.: Secure-by-construction synthesis of cyber-physical systems. Annual Reviews in Control **53**, 30–50 (2022). doi: https://doi.org/10.1016/j.arcontrol.2022.03.004 24. Moor, T., Davoren, J.M.: Robust controller synthesis for hybrid systems using modal logic. In: Benedetto, M.D.D., Sangiovanni-Vincentelli, A.L. (eds.) HSCC. LNCS, vol. 2034, pp. 433–446. Springer (2001)	**Controller *Envelope* Synthesis** • Bounds permissible controllers • Permits separation of safety critical and secondary concerns • Can be used, e.g., as trusted envelope for machine learning
Numerical Safety Shields 1. Safe Reinforcement Learning via Shielding, Alshiekh et al, AAAI 2018 2. Safe Reinforcement Learning via Formal Methods, Fulton et al, AAAI 2018 3. ModelPlex: Verified Runtime Validation of Verified Cyber-Physical System Model, RV 2014	**Symbolic** • Good for high dimension, infinite space/time problems • Statically computable
Manual Verified Design Case Studies 1. Platzer, A., Quesel, J.: European train control system: A case study in formal verification. In: Formal Methods and Software Engineering, 11th International Conference on Formal Engineering Methods, ICFEM 2009, Rio de Janeiro, Brazil, December 9-12, 2009.	**Automated** • Faster • Potentially more scalable for complex problems

Related work

Other Work	This Work
Controller Synthesis Techniques 7. Belta, C., Yordanov, B., Gol, E.A.: Formal Methods for Discrete-Time Dynamical Systems. Springer Cham (2017) 21. Liu, S., Trivedi, A., Yin, X., Zamani, M.: Secure-by-construction synthesis of cyber-physical systems. Annual Reviews in Control **53**, 30–50 (2022). doi: https://doi.org/10.1016/j.arcontrol.2022.03.004 24. Moor, T., Davoren, J.M.: Robust controller synthesis for hybrid systems using modal logic. In: Benedetto, M.D.D., Sangiovanni-Vincentelli, A.L. (eds.) HSCC. LNCS, vol. 2034, pp. 433–446. Springer (2001)	**Controller *Envelope* Synthesis** • Bounds permissible controllers • Permits separation of safety critical and secondary concerns • Can be used, e.g., as trusted envelope for machine learning
Numerical Safety Shields 1. Safe Reinforcement Learning via Shielding, Alshiekh et al, AAAI 2018 2. Safe Reinforcement Learning via Formal Methods, Fulton et al, AAAI 2018 3. ModelPlex: Verified Runtime Validation of Verified Cyber-Physical System Model, RV 2014	**Symbolic** • Good for high dimension, infinite space/time problems • Statically computable
Manual Verified Design Case Studies 1. Platzer, A., Quesel, J.: European train control system: A case study in formal verification. In: Formal Methods and Software Engineering, 11th International Conference on Formal Engineering Methods, ICFEM 2009, Rio de Janeiro, Brazil, December 9-12, 2009.	**Automated** • Faster • Potentially more scalable for complex problems

Related work

Other Work	This Work
Controller Synthesis Techniques 7. Belta, C., Yordanov, B., Gol, E.A.: Formal Methods for Discrete-Time Dynamical Systems. Springer Cham (2017) 21. Liu, S., Trivedi, A., Yin, X., Zamani, M.: Secure-by-construction synthesis of cyber-physical systems. Annual Reviews in Control **53**, 30–50 (2022). doi: https://doi.org/10.1016/j.arcontrol.2022.03.004 24. Moor, T., Davoren, J.M.: Robust controller synthesis for hybrid systems using modal logic. In: Benedetto, M.D.D., Sangiovanni-Vincentelli, A.L. (eds.) HSCC. LNCS, vol. 2034, pp. 433–446. Springer (2001)	**Controller *Envelope* Synthesis** • Bounds permissible controllers • Permits separation of safety critical and secondary concerns • Can be used, e.g., as trusted envelope for machine learning
Numerical Safety Shields 1. Safe Reinforcement Learning via Shielding, Alshiekh et al, AAAI 2018 2. Safe Reinforcement Learning via Formal Methods, Fulton et al, AAAI 2018 3. ModelPlex: Verified Runtime Validation of Verified Cyber-Physical System Model, RV 2014	**Symbolic** • Good for high dimension, infinite space/time problems • Statically computable
Manual Verified Design Case Studies 1. Platzer, A., Quesel, J.: European train control system: A case study in formal verification. In: Formal Methods and Software Engineering, 11th International Conference on Formal Engineering Methods, ICFEM 2009, Rio de Janeiro, Brazil, December 9-12, 2009.	**Automated** • Faster • Potentially more scalable for complex problems

Related work

Other Work	This Work
Controller Synthesis Techniques 7. Belta, C., Yordanov, B., Gol, E.A.: Formal Methods for Discrete-Time Dynamical Systems. Springer Cham (2017) 21. Liu, S., Trivedi, A., Yin, X., Zamani, M.: Secure-by-construction synthesis of cyber-physical systems. Annual Reviews in Control **53**, 30–50 (2022). doi: `https://doi.org/10.1016/j.arcontrol.2022.03.004` 24. Moor, T., Davoren, J.M.: Robust controller synthesis for hybrid systems using modal logic. In: Benedetto, M.D.D., Sangiovanni-Vincentelli, A.L. (eds.) HSCC. LNCS, vol. 2034, pp. 433–446. Springer (2001)	**Controller *Envelope* Synthesis** • Bounds permissible controllers • Permits separation of safety critical and secondary concerns • Can be used, e.g., as trusted envelope for machine learning
Numerical Safety Shields 1. Safe Reinforcement Learning via Shielding, Alshiekh et al, AAAI 2018 2. Safe Reinforcement Learning via Formal Methods, Fulton et al, AAAI 2018 3. ModelPlex: Verified Runtime Validation of Verified Cyber-Physical System Model, RV 2014	**Symbolic** • Good for high dimension, infinite space/time problems • Statically computable
Manual Verified Design Case Studies 1. Platzer, A., Quesel, J.: European train control system: A case study in formal verification. In: Formal Methods and Software Engineering, 11th International Conference on Formal Engineering Methods, ICFEM 2009, Rio de Janeiro, Brazil, December 9-12, 2009.	**Automated** • Faster • Potentially more scalable for complex problems

Overview

Part 2: Synthesis

- Introduction
- **Problem Statement**
- Game Logic and Solution
- Refinement
- Evaluation

Problem

Fill in holes ($\sqcup$) in a template with a propositional formula.

$$\mathsf{prob} \;\equiv\; \mathsf{assum} \wedge \sqcup \rightarrow \left[\left(\left(\cup_i \left(?\sqcup_i \,;\, \mathsf{act}_i\right)\right) ; \mathsf{plant}\right)^*\right] \mathsf{safe}.$$

Problem

Fill in holes ($\llcorner\!\lrcorner$) in a template with a propositional formula.

$$\mathsf{prob} \equiv \mathsf{assum} \land \llcorner\!\lrcorner \rightarrow [((\cup_i (?\llcorner\!\lrcorner_i ; \mathsf{act}_i)) ; \mathsf{plant})^*]\, \mathsf{safe}.$$

Initial Assumptions

Problem

Fill in holes ($\sqcup$) in a template with a propositional formula.

$$\mathsf{prob} \equiv \mathsf{assum} \wedge \sqcup \rightarrow [((\cup_i (?\sqcup_i ; \mathsf{act}_i)) ; \mathsf{plant})^*]\, \mathsf{safe}.$$

Control Loop

Initial Assumptions

Problem

Fill in holes ($\sqcup$) in a template with a propositional formula.

$$\mathsf{prob} \equiv \mathsf{assum} \land \sqcup \rightarrow \left[\left(\left(\cup_i \left(?\sqcup_i \,;\, \mathsf{act}_i\right)\right) ;\, \mathsf{plant}\right)^*\right] \mathsf{safe}.$$

Control Loop

Safety Contract

Initial Assumptions

Problem

Fill in holes ($\sqcup$) in a template with a propositional formula.

$$\text{prob} \equiv \text{assum} \wedge \sqcup \rightarrow \left[\left(\left(\cup_i \left(?\sqcup_i \,;\, \text{act}_i\right)\right) ; \text{plant}\right)^*\right] \text{safe}.$$

Problem

Fill in holes ($\sqcup$) in a template with a propositional formula.

$$\mathsf{prob} \equiv \mathsf{assum} \wedge \sqcup \rightarrow [((\cup_i (?\sqcup_i ; \mathsf{act}_i)) ; \mathsf{plant})^*]\, \mathsf{safe}.$$

Assumptions on the system

Problem

Fill in holes ($\sqcup$) in a template with a propositional formula.

Conditions for controllability

$$\mathsf{prob} \equiv \mathsf{assum} \wedge \sqcup \rightarrow [((\cup_i (?\sqcup_i ; \mathsf{act}_i)) ; \mathsf{plant})^*]\, \mathsf{safe}.$$

Assumptions on the system

Problem

Fill in holes ($\sqcup$) in a template with a propositional formula.

Conditions for controllability

$$\mathsf{prob} \equiv \mathsf{assum} \land \sqcup \rightarrow [((\cup_i (?\sqcup_i ; \mathsf{act}_i)) ; \mathsf{plant})^*]\, \mathsf{safe}.$$

Branch between i possible actions

Assumptions on the system

Problem

Fill in holes ($\sqcup$) in a template with a propositional formula.

Conditions for controllability

When is it ok to take action i?

$$\text{prob} \equiv \text{assum} \wedge \sqcup \rightarrow [((\cup_i (?\sqcup_i \, ; \, \text{act}_i)) \, ; \, \text{plant})^*] \, \text{safe}.$$

Branch between i possible actions

Assumptions on the system

Problem

Fill in holes ($\sqcup$) in a template with a propositional formula.

Conditions for controllability

When is it ok to take action i?

$$\text{prob} \equiv \text{assum} \land \sqcup \rightarrow [((\cup_i (?\sqcup_i ; \text{act}_i)) ; \text{plant})^*]\, \text{safe}.$$

Branch between i possible actions

Assumptions on the system

Physical environment

Problem

Fill in holes (⌴) in a template with a propositional formula.

Example:

Model 1 The train ETCS model (slightly modified from [29]). Framed parts can be automatically synthesized by our proposed tool.

Problem

Fill in holes (⌴) in a template with a propositional formula.

Example:

Model 1 The train ETCS model (slightly modified from [29]). Framed parts can be automatically synthesized by our proposed tool.

assum	1	$A > 0 \land B > 0 \land T > 0 \land v \geq 0$	Assumptions on the system

Problem

Fill in holes ($\lfloor\;\rfloor$) in a template with a propositional formula.

Example:

Model 1 The train ETCS model (slightly modified from [29]). Framed parts can be automatically synthesized by our proposed tool.

assum	1	$A > 0 \wedge B > 0 \wedge T > 0 \wedge v \geq 0$
ctrlable	2	$\wedge \boxed{} \rightarrow [\{$

Conditions from necessary to safety

Problem

Fill in holes (⎵) in a template with a propositional formula.

Example:

Model 1 The train ETCS model (slightly modified from [29]). Framed parts can be automatically synthesized by our proposed tool.

assum	1	$A > 0 \wedge B > 0 \wedge T > 0 \wedge v \geq 0$
ctrlable	2	$\wedge$ [ ] $\rightarrow [\{$
	3	((? [ ] ; $a := A$)

When is it ok to accelerate?

Problem

Fill in holes ($\sqcup$) in a template with a propositional formula.

Example:

Model 1 The train ETCS model (slightly modified from [29]). Framed parts can be automatically synthesized by our proposed tool.

assum	1	$A > 0 \land B > 0 \land T > 0 \land v \geq 0$
ctrlable	2	$\land \boxed{} \rightarrow [\{$
ctrl	3	$(\quad (?\boxed{}; a := A)$
	4	$\cup \ (?\boxed{}; a := -B) \quad);$

When is it ok to brake?

Problem

Fill in holes ($\sqcup$) in a template with a propositional formula.

Example:

Model 1 The train ETCS model (slightly modified from [29]). Framed parts can be automatically synthesized by our proposed tool.

assum	1	$A > 0 \wedge B > 0 \wedge T > 0 \wedge v \geq 0$
ctrlable	2	$\wedge \boxed{\quad\quad\quad} \rightarrow [\{$
ctrl	3	$(\quad (?\boxed{\quad\quad\quad\quad\quad\quad}; a := A)$
	4	$\cup\ (?\boxed{\quad}; a := -B)\quad);$
plant	5	$(t := 0; \{p' = v, v' = a, t' = 1 \ \& \ t \leq T \wedge v \geq 0\})$

System differential equation

Problem

Fill in holes ($\sqcup$) in a template with a propositional formula.

Example:

Model 1 The train ETCS model (slightly modified from [29]). Framed parts can be automatically synthesized by our proposed tool.

assum	$A > 0 \wedge B > 0 \wedge T > 0 \wedge v \geq 0$
ctrlable	$\wedge \boxed{} \rightarrow [\{$
ctrl	$(\quad (?\boxed{}; a := A)$
	$\cup\ (?\boxed{}; a := -B)\quad);$
plant	$(t := 0; \{p' = v, v' = a, t' = 1 \,\&\, t \leq T \wedge v \geq 0\})$
safe	$\}^*](e - p > 0)$ Safety contract

Problem: Example Solution

Fill in holes ($\sqcup$) in a template.

Example:

Model 1 The train ETCS model (slightly modified from [29]). Framed parts can be automatically synthesized by our proposed tool.

assum	$A > 0 \land B > 0 \land T > 0 \land v \geq 0$
ctrlable	$\land$ $\boxed{e - p > v^2/2B}$ $\rightarrow [\{$ — There's enough space to stop if we start braking now
ctrl	$(\quad (?\boxed{}; a := A)$
	$\cup\ (?\boxed{}; a := -B) \quad);$
plant	$(t := 0; \{p' = v, v' = a, t' = 1 \ \&\ t \leq T \land v \geq 0\})$
safe	$\}^*](e - p > 0)$

Problem: Example Solution

Fill in holes (⎵) in a template.

Example:

Model 1 The train ETCS model (slightly modified from [29]). Framed parts can be automatically synthesized by our proposed tool.

assum	1	$A > 0 \wedge B > 0 \wedge T > 0 \wedge v \geq 0$
ctrlable	2	$\wedge \boxed{e - p > v^2/2B} \rightarrow [\{$
	3	$(\quad (?\boxed{e - p > vT + AT^2/2 + (v + AT)^2/2B}; a := A)$
safe	6	$\}^*](e - p > 0)$

There's enough space to stop if we accelerate for one time period and then keep braking

Problem: Example Solution

Fill in holes ($\sqcup$) in a template.

Example:

Model 1 The train ETCS model (slightly modified from [29]). Framed parts can be automatically synthesized by our proposed tool.

assum	$A > 0 \wedge B > 0 \wedge T > 0 \wedge v \geq 0$
ctrlable	$\wedge \boxed{e - p > v^2/2B} \rightarrow [\{$
ctrl	$(\quad (?\boxed{e - p > vT + AT^2/2 + (v + AT)^2/2B}; a := A)$
	$\cup\ (?\boxed{\text{true}}; a := -B)\quad);$
plant	$(t := 0; \{p' = v, v' = a, t' = 1\ \&\ t \leq T \wedge v \geq 0\})$
safe	$\}^*](e - p > 0)$

You never make life worse by braking

Solution

Fill in holes ($\sqcup$) in a template with a propositional formula.

$$\mathsf{prob} \equiv \mathsf{assum} \wedge \sqcup \rightarrow [((\cup_i (?\sqcup_i ; \mathsf{act}_i)) ; \mathsf{plant})^*]\, \mathsf{safe}.$$

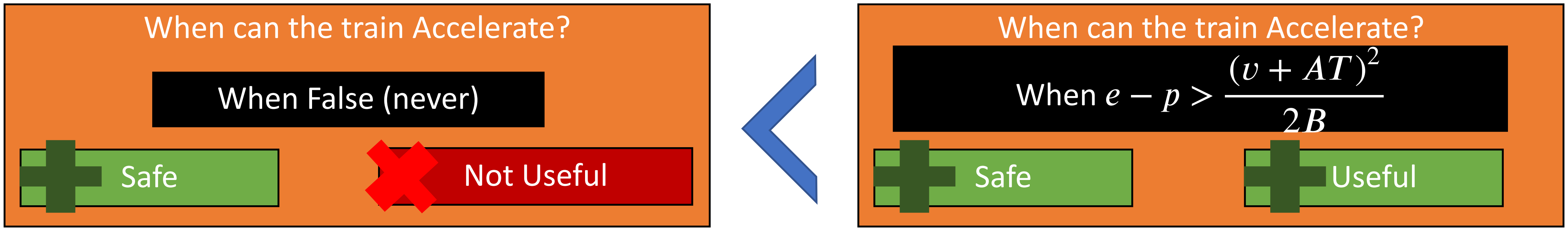
Solution

Fill in holes ($\llcorner\!\lrcorner$) in a template with a propositional formula.

$$\mathsf{prob} \equiv \mathsf{assum} \wedge \boxed{\llcorner I \lrcorner} \rightarrow \left[\left(\left(\cup_i \left(\boxed{? \llcorner G \lrcorner_i} \,;\, \mathsf{act}_i\right)\right) ; \mathsf{plant}\right)^* \right] \mathsf{safe}.$$

1. Safety (valid dL formula)
2. Always some control option $((\mathsf{assum} \wedge I) \rightarrow \vee_i \; G_i)$

Quality of Solution



- Good solution: more permissive
- $S' \geq S$ when $\vDash \text{assum} \rightarrow (I \rightarrow I')$ and $\vDash (\text{assum} \land I) \rightarrow \land_i (G_i \rightarrow G_i')$
- Unique optimum

Overview

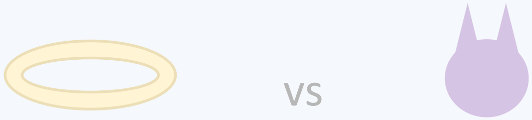
Part 2: Synthesis

- Introduction
- Problem Statement
- **Game Logic and Solution**
- Refinement
- Evaluation

Background: Game Logic

dL has nondeterminism
$(a := A \cup a := B)$

Players resolve nondeterminism



Operators

$(a := A \cup a := B)$

$(a := A \cap a := B)$

$\alpha \cap \beta,\ \alpha^{\times},\ ?\phi^{d},\ \{x' = f(x)Q\}^{d}$

Angelic Game

$\langle(a{:=}A \cap a{:=}B)\rangle a{=}A$

Angel wins if in the end, a=A

Demonic Game

$[(a := A \cap a := B)]a = A$

Demon wins if in the end, a=A

Duality

$\neg\langle\alpha\rangle\neg P \quad \leftrightarrow \quad [a]P$

Axioms

dGl without loops: translation in first order logic.

$[(v{:=}1 \cap v{:=}{-1});\{x^{\wedge\prime}=v\}]x \neq 0$

$\equiv [(v{:=}1)][\{x^{\wedge\prime}=v\}]x \neq 0 \vee [(v{:=}{-1})][\{x^{\wedge\prime}=v\}]x \neq 0$

$\equiv [\{x^{\wedge\prime}=1\}]x \neq 0 \vee [\{x^{\wedge\prime}=-1\}]x \neq 0$

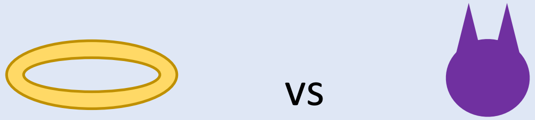
$\equiv \forall t \geq 0\, x+t \neq 0 \vee \forall t \geq 0\, x-t \neq 0$

$\equiv x>0 \vee x<0$

Background: Game Logic

dL has nondeterminism
$(a := A \cup a := B)$

Players resolve nondeterminism



Operators

$(a := A \cup a := B)$

$(a := A \cap a := B)$

$\alpha \cap \beta,\ \alpha^{\times},\ ?\phi^{d},\ \{x' = f(x)Q\}^{d}$

Angelic Game

⟨(a:=A∩a:=B)⟩a=A

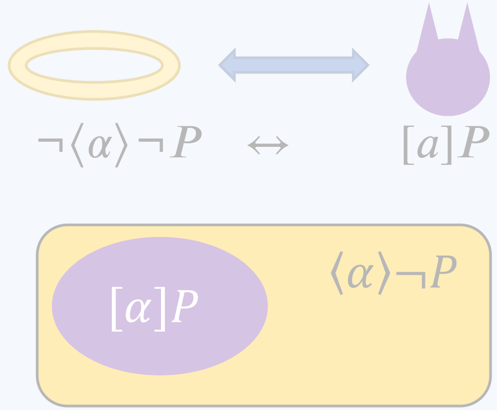
Angel wins if in the end, a=A

Demonic Game

$[(a := A \cap a := B)]a = A$

Demon wins if in the end, a=A

Duality



Axioms

dGl without loops: translation in first order logic.

[(v:=1∩v:=-1);{x^′=v}]x≠0

≡[(v:=1)][{x^′=v}]x≠0∨
[(v:=-1)][{x^′=v}]x≠0

≡[{x^′=1}]x≠0∨[{x^′=-1}]x≠0

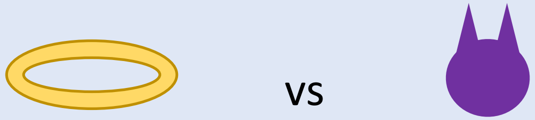
≡∀t≥0x+t≠0∨∀t≥0x-t≠0

≡x>0∨x<0

Background: Game Logic

dL has nondeterminism
$(a := A \cup a := B)$

Players resolve nondeterminism



vs

Operators

$(a := A \cup a := B)$

$(a := A \cap a := B)$

$\alpha \cap \beta,\ \alpha^{\times}, ?\phi^{d}\ ,\ \{x' = f(x)Q\}^{d}$

Angelic Game

$\langle(a:=A \cap a:=B)\rangle a=A$

Angel wins if in the end, a=A

Demonic Game

$[(a := A \cap a := B)]a = A$

Demon wins if in the end, a=A

Duality

$\neg\langle\alpha\rangle\neg P \quad \leftrightarrow \quad [\alpha]P$

Axioms

dGl without loops: translation in first order logic.

$[(v:=1 \cap v:=-1);\{x'=v\}]x\neq0$

$\equiv [(v:=1)][\{x'=v\}]x\neq0 \vee [(v:=-1)][\{x'=v\}]x\neq0$

$\equiv [\{x'=1\}]x\neq0 \vee [\{x'=-1\}]x\neq0$

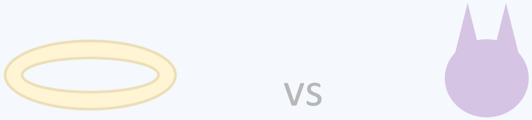
$\equiv \forall t\geq0\, x+t\neq0 \vee \forall t\geq0\, x-t\neq0$

$\equiv x>0 \vee x<0$

Background: Game Logic

dL has nondeterminism
$(a := A \cup a := B)$

Players resolve nondeterminism



Operators

$(a := A \cup a := B)$

$(a := A \cap a := B)$

$\alpha \cap \beta,\ \alpha^{\times},\ ?\phi^{d},\ \{x' = f(x)Q\}^{d}$

Angelic Game

⟨(a≔A∩a≔B)⟩a=A

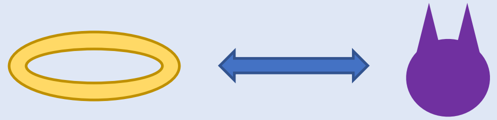
Angel wins if in the end, a=A

Demonic Game

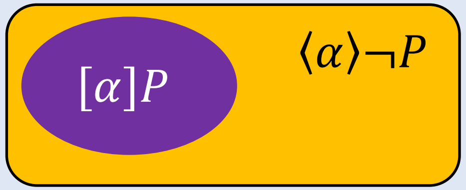
$[(a := A \cap a := B)]a = A$

Demon wins if in the end, a=A

Duality



$\neg\langle\alpha\rangle\neg P \quad \leftrightarrow \quad [a]P$



Axioms

dGl without loops: translation in first order logic.

[(v≔1∩v≔-1);{x^′=v}]x≠0

≡[(v≔1)][{x^′=v}]x≠0∨
[(v≔-1)][{x^′=v}]x≠0

≡[{x^′=1}]x≠0∨[{x^′=-1}]x≠0

≡∀t≥0x+t≠0∨∀t≥0x-t≠0

≡x>0∨x<0

Background: Game Logic

dL has nondeterminism
$(a := A \cup a := B)$

Players resolve nondeterminism

vs

Operators

$(a := A \cup a := B)$

$(a := A \cap a := B)$

$\alpha \cap \beta,\ \alpha^{\times},\ ?\phi^{d},\ \{x' = f(x)Q\}^{d}$

Angelic Game

$\langle(a{:=}A \cap a{:=}B)\rangle a{=}A$

Angel wins if in the end, a=A

Demonic Game

$[(a := A \cap a := B)]a = A$

Demon wins if in the end, a=A

Duality

$\neg\langle\alpha\rangle\neg P \quad \leftrightarrow \quad [a]P$

Axioms

dGl without loops: translation in first order logic.

$[(v{:=}1 \cap v{:=}{-1});\{x' = v\}]x \neq 0$

$\equiv [(v{:=}1)][\{x' = v\}]x \neq 0 \lor [(v{:=}{-1})][\{x' = v\}]x \neq 0$

$\equiv [\{x' = 1\}]x \neq 0 \lor [\{x' = -1\}]x \neq 0$

$\equiv \forall t \geq 0\, x+t \neq 0 \lor \forall t \geq 0\, x-t \neq 0$

$\equiv x > 0 \lor x < 0$

Optimal Solution

$$\text{prob} \;\equiv\; \text{assum} \wedge \;\sqcup\; \rightarrow [((\cup_i\, (?\sqcup_i\, ;\, \text{act}_i))\, ;\, \text{plant})^*]\, \text{safe}.$$

The set of all states from which a perfect controller can keep the system safe forever

$$I^{\text{opt}} \;\equiv\; [((\cap_i\, \text{act}_i)\, ;\, \text{plant})^*]\, \text{safe}$$

Controller chooses in its best interest

By construction, loop invariant

Optimal Solution

$$\text{prob} \equiv \text{assum} \wedge \sqcup \rightarrow [((\cup_i (?\sqcup_i ; \text{act}_i)) ; \text{plant})^*] \text{safe}.$$

The set of all states from which a perfect controller can keep the system safe forever

$$I^{\text{opt}} \equiv [((\cap_i \text{act}_i) ; \text{plant})^*] \text{safe}$$

Controller chooses in its best interest

By construction, loop invariant

Allow any control action that is guaranteed to keep the system within I^{opt}

$$G_i^{\text{opt}} \equiv [\text{act}_i ; \text{plant}] I^{\text{opt}}.$$

Computing Propositional Arithmetic Solutions

- Easily checked at runtime

Computing Propositional Arithmetic Solutions

- Easily checked at runtime
- Use the semantics of dGL (which are in terms of FOL^*)

Computing Propositional Arithmetic Solutions

- Easily checked at runtime
- Use the semantics of dGL (which are in terms of FOL^*)
- *But two dGL constructions need more than FOL.

Computing Propositional Arithmetic Solutions

- Easily checked at runtime
- Use the semantics of dGL (which are in terms of FOL^*)
- *But two dGL constructions need more than FOL.

  - Loops: Defined in terms of fixed point → Approximate with "Refinement"
  - Differential equations: Presupposes an ODE solution → Approximate

Overview

Part 2: Synthesis

- Introduction
- Problem Statement
- Game Logic and Solution
- **Refinement**
- Evaluation

Refinement

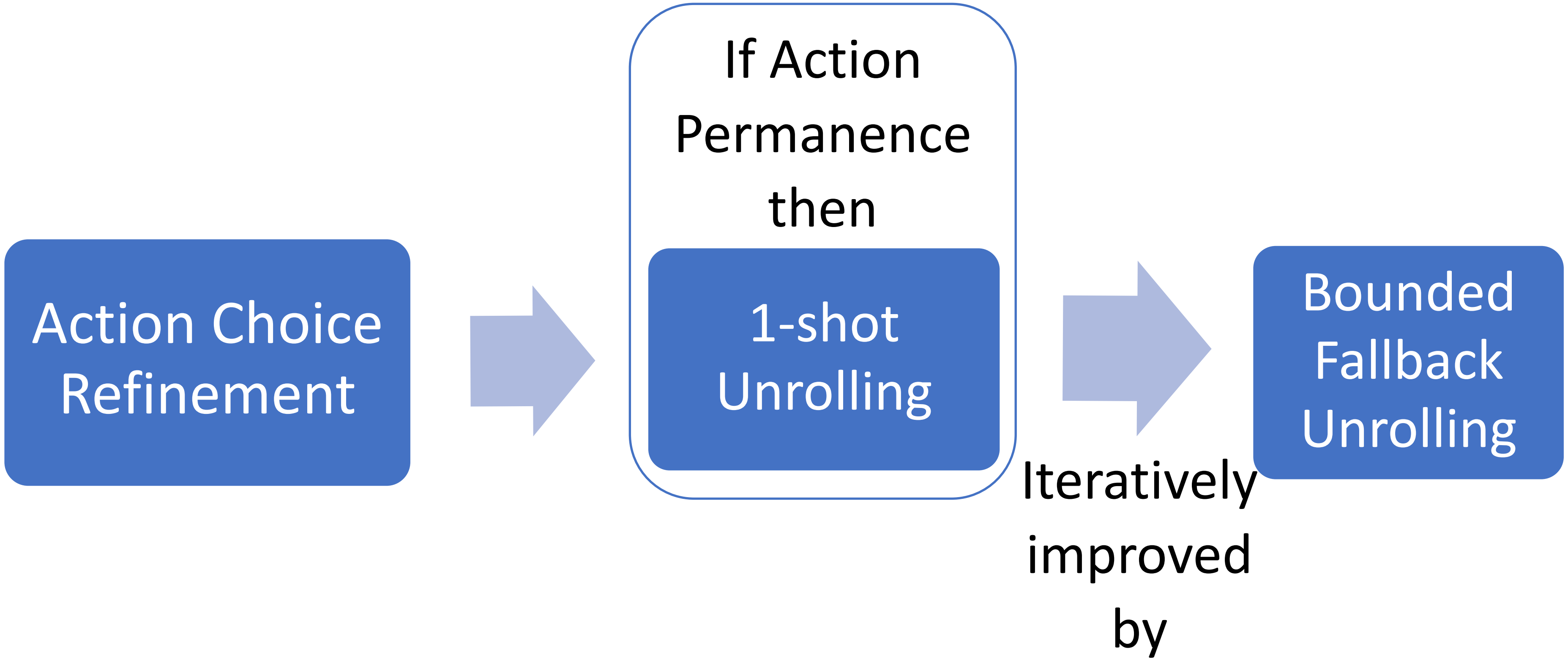
$$I^{\text{opt}} \equiv [((\cap_i \mathsf{act}_i) ; \mathsf{plant})^*] \mathsf{safe}$$

Want to remove this

Refinement

$$I^{\text{opt}} \equiv [((\cap_i \text{act}_i)\,;\ \text{plant})^*]\ \text{safe}$$

Want to remove this



Action Choice Refinement

The game obtained by restricting the controller to one action

$$\left[\left(\boldsymbol{a} := -\boldsymbol{B}; t := 0; \left\{p' = v, v' = a, t' = 1 \ \ t \le T \wedge v \ge 0\right\}\right)^*\right] e - p > 0$$

Is harder than the game where the controller chooses between multiple actions

$$\left[\left((\boldsymbol{a} := -\boldsymbol{B} \cap \boldsymbol{a} := \boldsymbol{A}); t := 0; \left\{p' = v, v' = a, t' = 1 \ \ t \le T \wedge v \ge 0\right\}\right)^*\right] e - p > 0$$

Action Choice Refinement

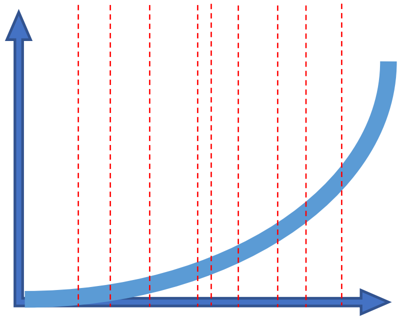
The game obtained by restricting the controller to one action

$$\left[\left(\boldsymbol{a} := -\boldsymbol{B}; t := 0; \left\{p' = v, v' = a, t' = 1 \;\; t \le T \wedge v \ge 0\right\}\right)^*\right] e - p > 0$$

Is harder than the game where the controller chooses between multiple actions

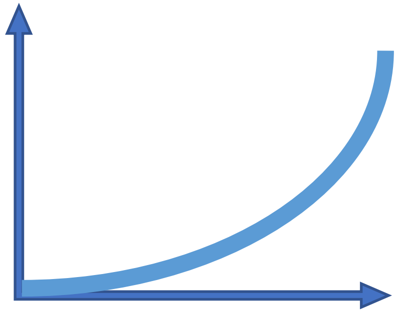
$$\left[\left((\boldsymbol{a} := -\boldsymbol{B} \cap \boldsymbol{a} := \boldsymbol{A}); t := 0; \left\{p' = v, v' = a, t' = 1 \;\; t \le T \wedge v \ge 0\right\}\right)^*\right] e - p > 0$$

One Shot Unrolling



If you repeat a time bounded ODE

$$\left[\left(a := -B; \boldsymbol{t := 0}; \left\{p' = v, v' = a, t' = 1 \;\; \boldsymbol{t \le T} \wedge v \ge 0\right\}\right)^*\right] e - p > 0$$



That's like executing the ODE for arbitrarily long

$$\left[a := -B; \left\{p' = v, v' = a, t' = 1 \;\; v \ge 0\right\}\right] e - p > 0$$

Action Choice Refinement

The game obtained by restricting the controller to one action

$$\left[\left(\boldsymbol{a := -B};\, t := 0;\; \left\{p' = v, v' = a, t' = 1 \;\; t \le T \wedge v \ge 0\right\}\right)^*\right] e - p > 0$$

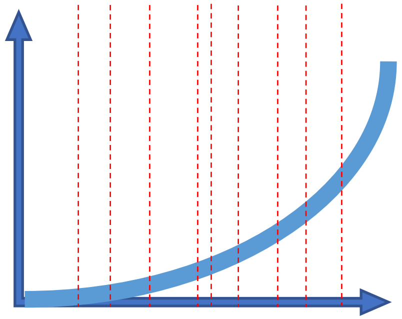
2

Is harder than the game where the controller chooses between multiple actions

$$\left[\left((\boldsymbol{a := -B} \cap \boldsymbol{a := A});\, t := 0;\; \left\{p' = v, v' = a, t' = 1 \;\; t \le T \wedge v \ge 0\right\}\right)^*\right] e - p > 0$$

1

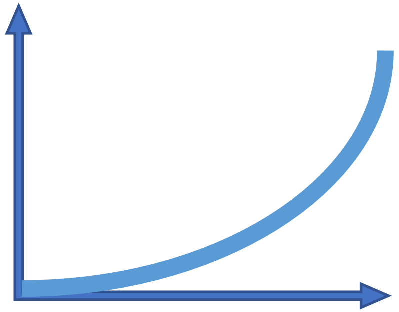
One Shot Unrolling



If you repeat a time bounded ODE

$$\left[\left(a := -B;\, \boldsymbol{t := 0};\; \left\{p' = v, v' = a, t' = 1 \;\; \boldsymbol{t \le T} \wedge v \ge 0\right\}\right)^*\right] e - p > 0$$

3



That's like executing the ODE for arbitrarily long

$$\left[a := -B;\; \left\{p' = v, v' = a, t' = 1 \;\; v \ge 0\right\}\right] e - p > 0$$

4

One Shot Refinement

$$\left[a := -B;\ \left\{p' = v, v' = a, t' = 1 \quad v \geq 0\right\}\right] e - p > 0$$

Symbolic Execution

$$\forall t\left(v - Bt \geq 0 \ \rightarrow p + vt - \frac{Bt^2}{2} > e\right)$$

Quantifier Elimination

$$I = \boxed{p + \frac{v^2}{2B} > e}$$

One Shot Refinement

$$\left[a := -B;\ \left\{p' = v, v' = a, t' = 1\ \ v \geq 0\right\}\right] e - p > 0$$

Symbolic Execution

$$\forall t \left(v - Bt \geq 0\ \rightarrow p + vt - \frac{Bt^2}{2} > e\right)$$

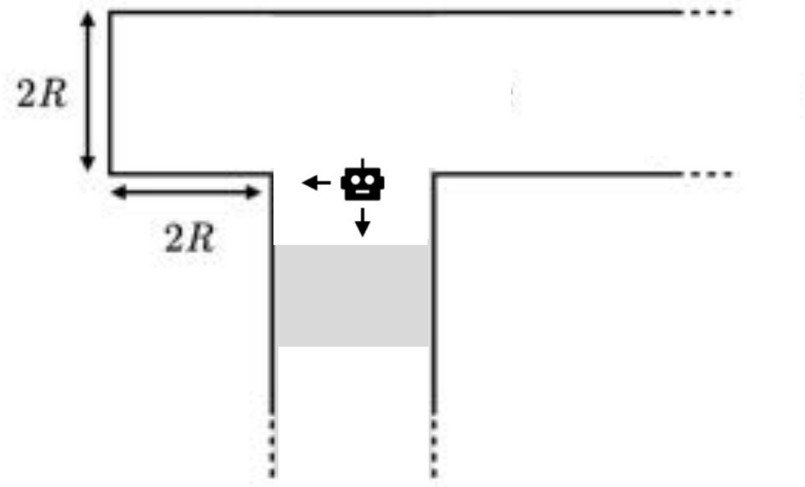
Quantifier Elimination

$$I = \boxed{p + \frac{v^2}{2B} > e}$$

- Action permanence: $(\mathsf{act}_i\ ;\ \mathsf{plant}\ ;\ \mathsf{act}_i) \equiv (\mathsf{act}_i\ ;\ \mathsf{plant})$
- In practice: when a control action corresponds to a “mode” of behavior.

One-shot Unrolling: Example

- 1-shot unrolling lets the controller choose one action and run it forever.



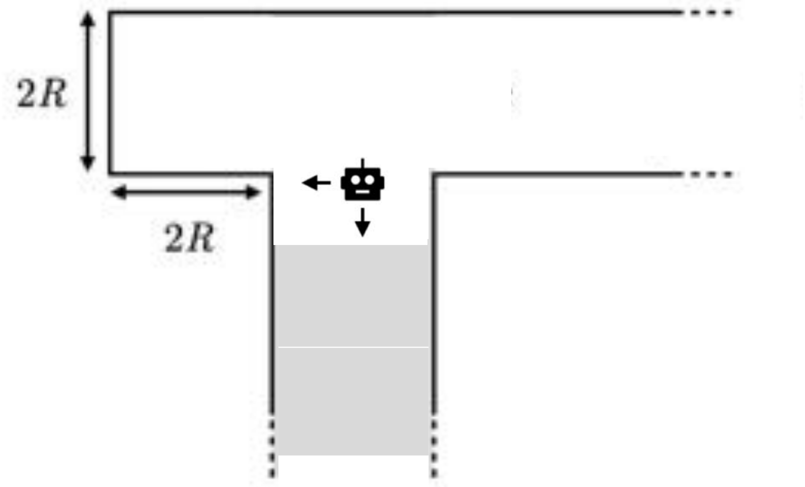
1 iteration

1-shot unroll

2-shot unroll

One-shot Unrolling: Example

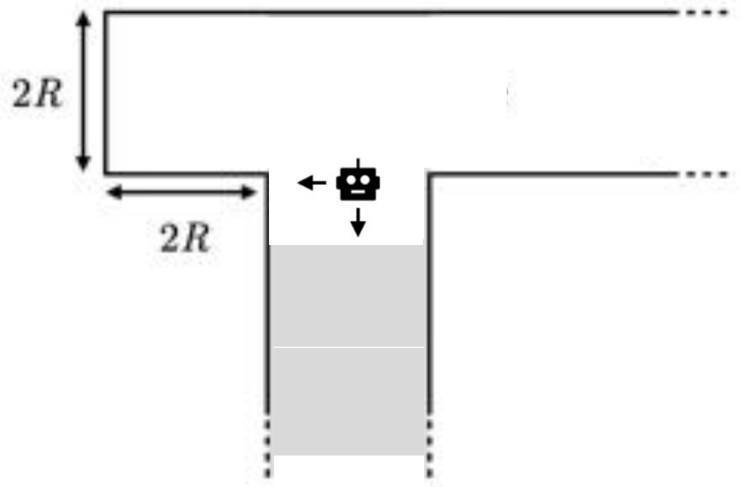
- 1-shot unrolling lets the controller choose one action and run it forever.

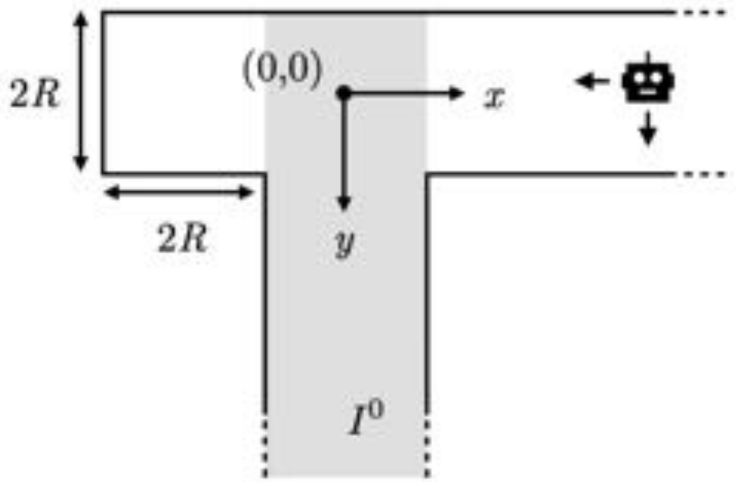


2 iterations 1-shot unroll 2-shot unroll

One-shot Unrolling: Example

- 1-shot unrolling lets the controller choose one action and run it forever.





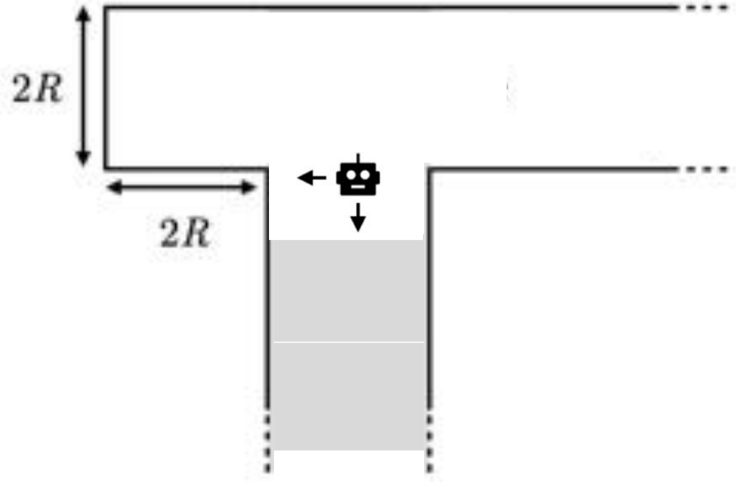
2 iterations

1-shot unroll

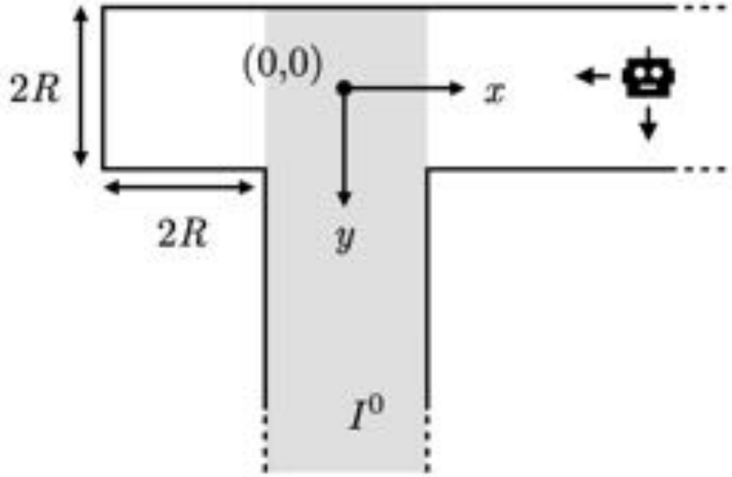
2-shot unroll

One-shot Unrolling: Example

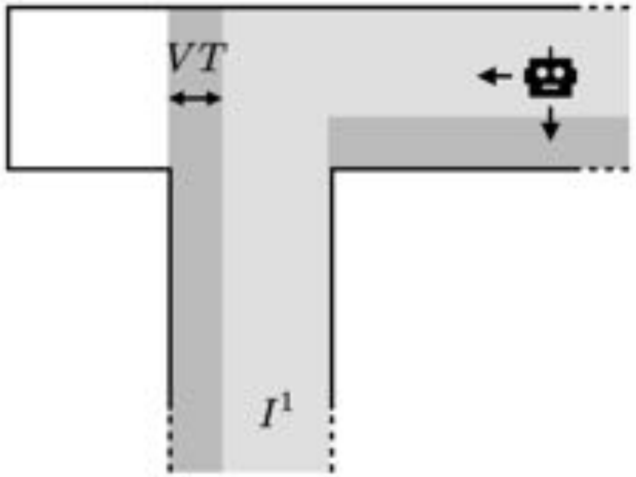
- 1-shot unrolling lets the controller choose one action and run it forever.



2 iterations



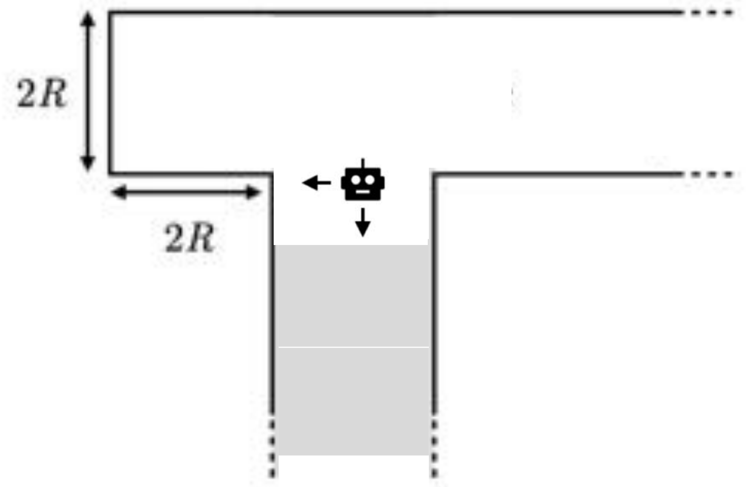
1-shot unroll



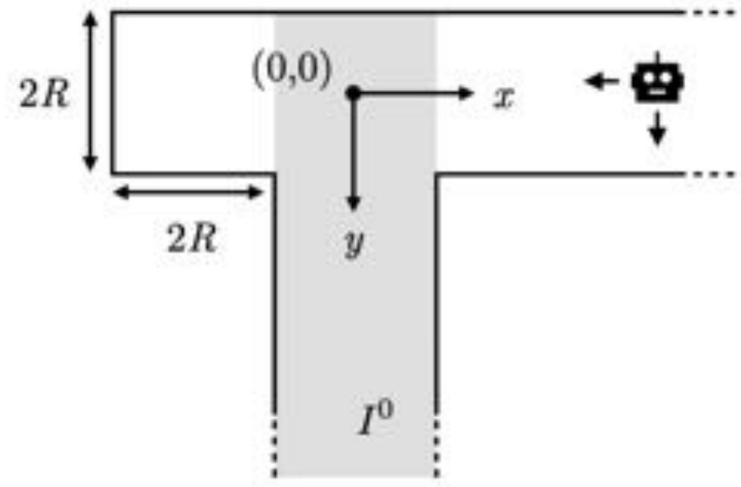
2-shot unroll

One-shot Unrolling: Example

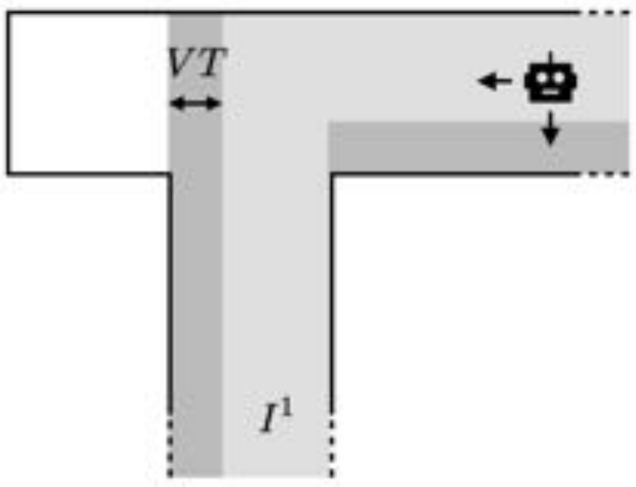
- 1-shot unrolling lets the controller choose one action and run it forever.
- Bounded unrolling allows a “switch” in action choice



2 iterations



1-shot unroll



2-shot unroll

Bounded Unrolling

- n switches to reach the region I_o in which safety is guaranteed indefinitely
- Controller has chance to switch within $[\theta, \theta + T]$ window because plant can never execute for time greater than T

Bounded Unrolling

- n switches to reach the region I_o in which safety is guaranteed indefinitely
- Controller has chance to switch within $[\theta, \theta + T]$ window because plant can never execute for time greater than T

$$\text{forever} \equiv (\cap_{i \in \mathsf{P}}\ \text{act}_i)\ ;\ \text{plant}_\infty$$

$$\text{step} \equiv (\theta := *\ ;\ ?\theta \geq 0)^d\ ;\ (\cap_{i \in \mathsf{P}}\ \text{act}_i)\ ;\ \text{plant}_{\theta+T}\ ;\ ?\text{safe}^d\ ;\ ?t \geq \theta$$

Controller chooses some time θ in the future

For a controller choice chosen up to time θ

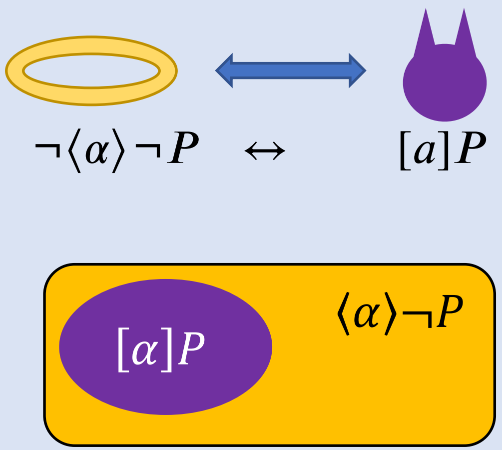
While staying safe

By time θ the controller reaches established safe region I^{n-1}

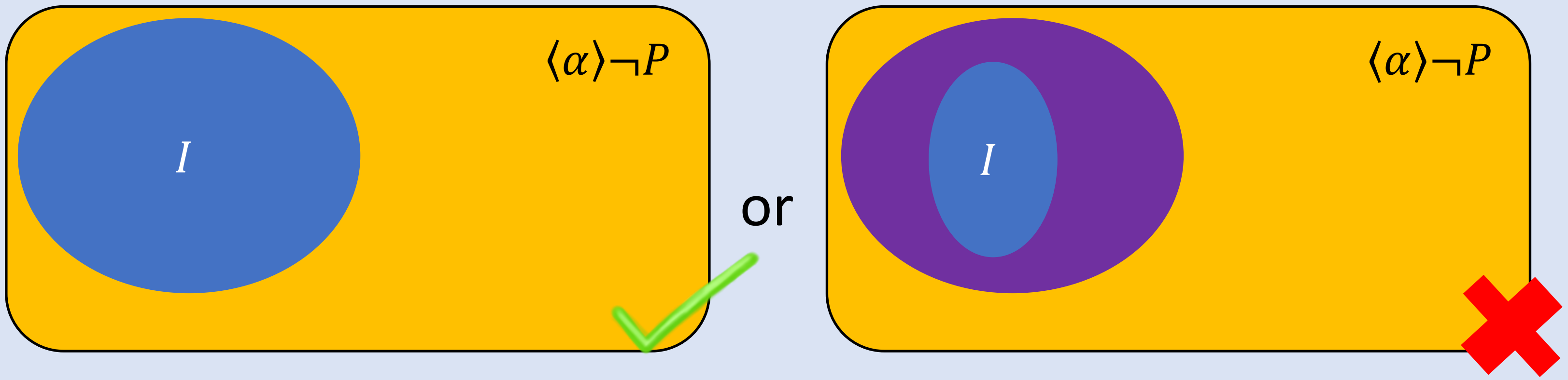
$$I^{n+1} \equiv I^n \vee [\text{step}]\, I^n \qquad I^0 \equiv [\text{forever}]\, \text{safe}$$

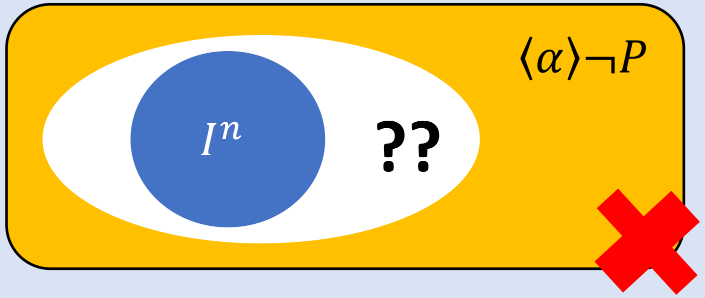
Dual Game

Duality



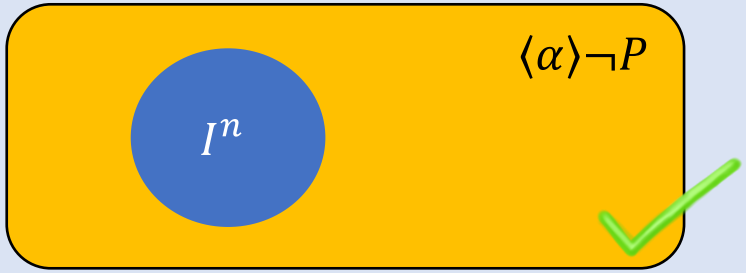
Optimal?





Check Environment Game

$(\langle\alpha\rangle\neg P)$



Algorithm: CESAR

- Recursively compute bounded unrolled invariants I^n.

Algorithm: CESAR

- Recursively compute bounded unrolled invariants I^n.
- Stop if either
  - Reached fixed point (I^n= I^{n+1})
  - Optimality check using dual succeeds (in all regions not in the invariant, can environment "win"?)
  - Unrolling budget reached

Algorithm: CESAR

- Recursively compute bounded unrolled invariants I^n.
- Stop if either
  - Reached fixed point ($I^n = I^{n+1}$)
  - Optimality check using dual succeeds (in all regions not in the invariant, can environment "win"?)
  - Unrolling budget reached
- With resulting I, compute each hole fill using

$$G_i \equiv [\text{act}_i\ ;\ \text{plant}]\, I$$

Overview

Part 2: Synthesis

- Introduction
- Problem Statement
- Game Logic and Solution
- Refinement
- **Evaluation**

Evaluation

Benchmark Suite with different control challenges

Table 2: Summary of the benchmark suite and most important control challenges.

Benchmark	Control Feature Introduced
Gears	Many (namely 8) actions to choose from.
ETCS	Nondeterministic, bounded acceleration (from case study [29]).
Table Tennis	Introduce two-dimensional motion.
Reservoir	Dynamics mixes variables that controller can and can't influence.
Reaction	Conjunctive safety constraints.
Merge	Disjunctive safety constraints.
Wall	Requires state-dependent fallback actions.
Parachute	Action switching restricted: cannot close parachute once open.
Corridor	Requires unrolling fallback for optimal synthesis (Fig. 1).
Sputtering Car	Unsolvable continuous dynamics.

Evaluation

Benchmark	Synthesis Time (s)	Memory (MB)	Checking Time (s)
Gears	5.97	41.30	2.6
ETCS	4.32	40.96	7.6
Table Tennis	2.79	40.13	1.4
Reservoir	4.95	39.99	2.1
Reaction	9.93	41.10	3.1
Merge	3.30	40.22	4.7
Wall	3.74	40.33	11.7
Parachute	3.37	40.16	5.0
Corridor*	7.14	41.71	1.9
Sputtering Car	2.12	39.63	1.1

Future Work

- Handle hard dynamics

Unknown Functions	Circular Dependencies	Taylor Polynomials	Ghost Dynamics

- Generalize to differential game logic

Time Triggered Control	Event Triggered Control	Free Assignments	Adversarial Agents

Thank You!

Aditi Kabra

https://aditink.github.io

akabra@cs.cmu.edu