Minimizing Sequents to Find Modeling Errors in KeYmaera X

By Ben Gafford and Myra Dotzel

Modeling mistakes happen!

Don't let them get the best of you.

Problem domain

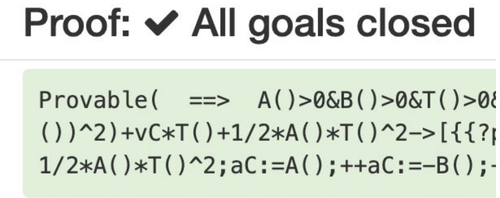
+

Model of controller and dynamics

```
ArchiveEntry "Time-triggered Adaptive Cruise Control"

Definitions
    Real A;   /* The controlled car accelerates with acceleration A */
    Real B;   /* The controlled car brakes with acceleration -B */
    Real T;   /* Maximal time between consecutive control cycles */
End.

  ProgramVariables
    Real pL;  /* Position of the lead car */
    Real vL;  /* Velocity of the lead car */
    Real aL;  /* Acceleration of the lead car */

    Real pC;  /* Position of the controlled car */
    Real vC;  /* Velocity of the controlled car */
    Real aC;  /* Acceleration of the controlled car */

    Real t;   /* Elapsed time since the last controller cycle */
End.

  Problem
    /* Initial conditions */
    ( A > 0 & B > 0 & T > 0 & vL >0 & pC <= pL &
      pL + (vL-B*T)*(-(vL-B*T)/(-B)) + 1/2 * -B * (-(vL-B*T)/(-B))^2 + vL * T - 1/2*B*T^2 >= pC + (vC + A*T)*(vC+A*T)/B + 1/2*-B*((vC+A*T)/B)^2 + vC*T + 1/2*A*T^2 )
    ->
    [
      {
        /* Controller for the controlled car */
        {
          {?pL + (vL-B*T)*(-(vL-B*T)/(-B)) + 1/2 * -B * (-(vL-B*T)/(-B))^2 + vL * T - 1/2*B*T^2 > pC + (vC + A*T)*(vC+A*T)/B + 1/2*-B*((vC+A*T)/B)^2 + vC*T + 1/2*A*T^2; aC := A;}
          ++{aC := -B;}
          ++{?vC=0;aC:=0;}
        }
        /* Controller for the lead car */
        {
          aL := A; ++ aL := -B; ++ {?vL=0;aL:=0;}
        }
        /* Continuous dynamics */
        {
          t := 0;
          { pL' = vL, vL' = aL, pC' = vC, vC' = aC, t'=1 &
            t<T & vL >= 0 & vC >= 0 }
        }
      }*@invariant(
        /* Loop invariant */
        A > 0 & B > 0 & T > 0 & pC <= pL &
        (vC >= vL -> pL + (vL)*(vL/B) + 1/2 * -B * (vL/B)^2 >= pC + (vC)*(vC)/B + 1/2*-B*((vC)/B)^2)
      )
    ]
    /* Safety condition */
    ( pC <= pL )
  End.
End.
```

+

Proof of safety properties



Proof: ✔ All goals closed

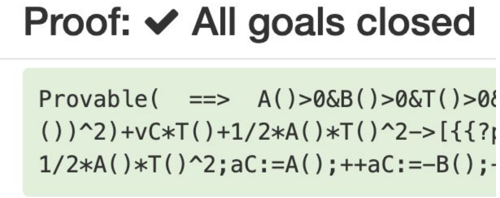
```
Provable(  ==>  A()>0&B()>0&T()>0&
())^2)+vC*T()+1/2*A()*T()^2->[{{?p
1/2*A()*T()^2;aC:=A();++aC:=-B();-
```

A model for adaptive cruise control

```
/* Initial conditions */
( A > 0 & B > 0 & vL >0 & pC < pL &
(vC>=vL -> (pL-pC > vC*(vL-vC)/-B + 1/2*-B*((vL-vC)/-B)^2 - (vL * ((vL-vC)/-B)))))
->
[
  {
    /* Control */
    {
      /* Safely assign acceleration or braking for the controlled car */
       {?(vC>=vL -> (pL-pC > vC*(vC-vL)/-B + 1/2*-B*((vC-vL)/-B)^2 - (vL * ((vC-vL)/-B)))); aC := A;}
       {?vC=0;aC := 0;}
       ++{aC := -B;}
    }
    /* Continuous dynamics and event triggers */
    {
      { pL' = vL, pC' = vC, vC' = aC &
        (vC>=vL -> (pL-pC <= vC*(vC-vL)/-B + 1/2*-B*((vC-vL)/-B)^2 - (vL * ((vC-vL)/-B)))) & vC >= 0 }
      ++
      { pL' = vL, pC' = vC, vC' = aC &
        (vC>=vL -> (pL-pC <= vC*(vC-vL)/-B + 1/2*-B*((vC-vL)/-B)^2 - (vL * ((vC-vL)/-B)))) & vC >= 0 }
    }
  }*
]
/* Safety condition */
( pC <= pL )
```

A model for ***provably safe*** adaptive cruise control!

```
/* Initial conditions */
( A > 0 & B > 0 & vL >0 & pC < pL &
(vC>=vL -> (pL-pC > vC*(vL-vC)/-B + 1/2*-B*((vL-vC)/-B)^2 - (vL * ((vL-vC)/-B)))))
->
[
  {
    /* Control */
    {
      /* Safely assign acceleration or braking for the controlled car */
       {?(vC>=vL -> (pL-pC > vC*(vC-vL)/-B + 1/2*-B*((vC-vL)/-B)^2 - (vL * ((vC-vL)/-
       {?vC=0;aC := 0;}
       ++{aC := -B;}
    }
    /* Continuous dynamics and event triggers */
    {
      { pL' = vL, pC' = vC, vC' = aC &
        (vC>=vL -> (pL-pC <= vC*(vC-vL)/-B + 1/2*-B*((vC-vL)/-B)^2 - (vL * ((vC-vL)/-B)))) & vC >= 0 }
      ++
      { pL' = vL, pC' = vC, vC' = aC &
        (vC>=vL -> (pL-pC <= vC*(vC-vL)/-B + 1/2*-B*((vC-vL)/-B)^2 - (vL * ((vC-vL)/-B)))) & vC >= 0 }
    }
  }*
]
/* Safety condition */
( pC <= pL )
```



Proof: ✔ All goals closed

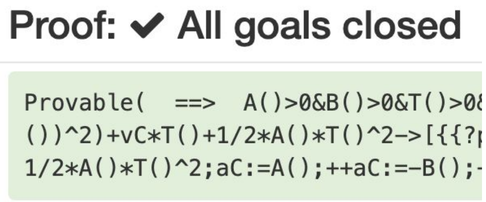
```
Provable(  ==>  A()>0&B()>0&T()>0
())^2)+vC*T()+1/2*A()*T()^2->[{{?
1/2*A()*T()^2;aC:=A();++aC:=-B();
```

…oh oops sorry one second

```
/* Initial conditions */
( A > 0 & B > 0 & vL >0 & pC < pL &
(vC>=vL -> (pL-pC > vC*(vL-vC)/-B + 1/2*-B*((vL-vC)/-B)^2 - (vL * ((vL-vC)/-B)))))
->
[
  {
    /* Control */
    {
      /* Safely assign acceleration or braking for the controlled car */
       {?(vC>=vL -> (pL-pC > vC*(vC-vL)/-B + 1/2*-B*((vC-vL)/-B)^2 - (vL * ((vC-vL)/-B)))); aC := A;}
       {?vC=0;aC := 0;}
       ++{aC := -B;}
    }
    /* Continuous dynamics and event triggers */
    {
      { pL' = vL, pC' = vC, vC' = aC &
        (vC>=vL -> (pL-pC <= vC*(vC-vL)/-B + 1/2*-B*((vC-vL)/-B)^2 - (vL * ((vC-vL)/-B)))) & vC >= 0 }
      ++
      { pL' = vL, pC' = vC, vC' = aC &
        (vC>=vL -> (pL-pC >= vC*(vC-vL)/-B + 1/2*-B*((vC-vL)/-B)^2 - (vL * ((vC-vL)/-B)))) & vC >= 0 }
    }
  }*
]
/* Safety condition */
( pC <= pL )
```

…now, a ***provably safe*** adaptive cruise control!

```
/* Initial conditions */
( A > 0 & B > 0 & vL >0 & pC < pL &
(vC>=vL -> (pL-pC > vC*(vL-vC)/-B + 1/2*-B*((vL-vC)/-B)^2 - (vL * ((vL-vC)/-B)))))
->
[
 {
  /* Control */
  {
   /* Safely assign acceleration or braking for the controlled car */
    {?(vC>=vL -> (pL-pC > vC*(vC-vL)/-B + 1/2*-B*((vC-vL)/-B)^2 - (vL * ((vC-vL)/-B))))
    {?vC=0;aC := 0;}
    ++{aC := -B;}
  }
  /* Continuous dynamics and event triggers */
  {
   { pL' = vL, pC' = vC, vC' = aC &
    (vC>=vL -> (pL-pC <= vC*(vC-vL)/-B + 1/2*-B*((vC-vL)/-B)^2 - (vL * ((vC-vL)/-B)))) & vC >= 0 }
   ++
   { pL' = vL, pC' = vC, vC' = aC &
    (vC>=vL -> (pL-pC >= vC*(vC-vL)/-B + 1/2*-B*((vC-vL)/-B)^2 - (vL * ((vC-vL)/-B)))) & vC >= 0 }
  }
 }*
]
/* Safety condition */
( pC <= pL )
```



Proof: ✔ All goals closed

```
Provable(  ==>  A()>0&B()>0&T()>0
())^2)+vC*T()+1/2*A()*T()^2->[{{?
1/2*A()*T()^2;aC:=A();++aC:=-B();
```

…sorry this is embarrassing

```
/* Initial conditions */
( A > 0 & B > 0 & vL >0 & pC < pL &
(vC>=vL -> (pL-pC > vC*(vL-vC)/-B + 1/2*-B*((vL-vC)/-B)^2 - (vL * ((vL-vC)/-B)))))
->
[
  {
    /* Control */
    {
      /* Safely assign acceleration or braking for the controlled car */
       {?(vC>=vL -> (pL-pC > vC*(vC-vL)/-B + 1/2*-B*((vC-vL)/-B)^2 - (vL * ((vC-vL)/-B)))); aC := A;}
       ++{?vC=0;aC := 0;}
       ++{aC := -B;}
    }
    /* Continuous dynamics and event triggers */
    {
      { pL' = vL, pC' = vC, vC' = aC &
        (vC>=vL -> (pL-pC <= vC*(vC-vL)/-B + 1/2*-B*((vC-vL)/-B)^2 - (vL * ((vC-vL)/-B))) & vC >= 0 }
      ++
      { pL' = vL, pC' = vC, vC' = aC &
        (vC>=vL -> (pL-pC >= vC*(vC-vL)/-B + 1/2*-B*((vC-vL)/-B)^2 - (vL * ((vC-vL)/-B))) & vC >= 0 }
    }
  }*
]
/* Safety condition */
( pC <= pL )
```

…sorry this is even more embarrassing

```
/* Initial conditions */
( A > 0 & B > 0 & vL >0 & pC < pL &
(vC>=vL -> (pL-pC > vC*(vL-vC)/-B + 1/2*-B*((vL-vC)/-B)^2 - (vL * ((vL-vC)/-B)))))
->
[
  {
    /* Control */
    {
      /* Safely assign acceleration or braking for the controlled car */
       {?(vC>=vL -> (pL-pC > vC*(vL-vC)/-B + 1/2*-B*((vL-vC)/-B)^2 - (vL * ((vL-vC)/-B)))); aC := A;}
       ++{?vC=0;aC := 0;}
       ++{aC := -B;}
    }
    /* Continuous dynamics and event triggers */
    {
      { pL' = vL, pC' = vC, vC' = aC &
        (vC>=vL -> (pL-pC <= vC*(vL-vC)/-B + 1/2*-B*((vL-vC)/-B)^2 - (vL * ((vL-vC)/-B))) & vC >= 0 }
      ++
      { pL' = vL, pC' = vC, vC' = aC &
        (vC>=vL -> (pL-pC >= vC*(vL-vC)/-B + 1/2*-B*((vL-vC)/-B)^2 - (vL * ((vL-vC)/-B))) & vC >= 0 }
    }
  }*
]
/* Safety condition */
( pC <= pL )
```

Motivation

- Problem
    - Difficult for programmers to catch their own errors
    - Over-constrained models often lead to weak models
    - Erroneous models are dangerous and can be expensive to cope with
- Our solution
    - A really good analysis.

How can we identify these over-constraining errors?

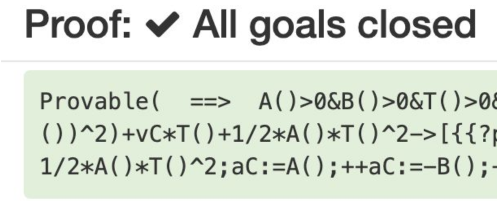
```
/* Initial conditions */
( A > 0 & B > 0 & vL >0 & pC < pL &
(vC>=vL -> (pL-pC > vC*(vL-vC)/-B + 1/2*-B*((vL-vC)/-B)^2 - (vL * ((vL-vC)/-B)))))
->
[
  {
    /* Control */
    {
      /* Safely assign acceleration or braking for the controlled car */
       {?(vC>=vL -> (pL-pC > vC*(vL-vC)/-B + 1/2*-B*((vL-vC)/-B)^2 - (vL * ((vL-vC)/-B)))); aC := A;}
       ++{?vC=0;aC := 0;}
       ++{aC := -B;}
    }
    /* Continuous dynamics and event triggers */
    {
      { pL' = vL, pC' = vC, vC' = aC &
        (vC>=vL -> (pL-pC <= vC*(vL-vC)/-B + 1/2*-B*((vL-vC)/-B)^2 - (vL * ((vL-vC)/-B)))) & vC >= 0 }
      ++
      { pL' = vL, pC' = vC, vC' = aC &
        (vC>=vL -> (pL-pC >= vC*(vL-vC)/-B + 1/2*-B*((vL-vC)/-B)^2 - (vL * ((vL-vC)/-B)))) & vC >= 0 }
    }
  }*
]
/* Safety condition */
( pC <= pL )
```

How can we identify these over-constraining errors?

```
/* Initial conditions */
( A > 0 & B > 0 & vL >0 & pC < pL &
(vC>=vL -> (pL-pC > vC*(vL-vC)/-B + 1/2*-B*((vL-vC)/-B)^2 - (vL * ((vL-vC)/-B)))))
->
[
  {
    /* Control */
    {
      /* Safely assign acceleration or braking for the controlled car */
       {?(vC>=vL -> (pL-pC > vC*(vL-vC)/-B + 1/2*-B*((vL-vC)/-B)^2 - (vL * ((vL-vC)/-B)))); aC := A;}
       ++{?vC=0;aC := 0;}
       ++{aC := -B;}
    }
    /* Continuous dynamics and event triggers */
    {
      { pL' = vL, pC' = vC, vC' = aC &
        (vC>=vL -> (pL-pC <= vC*(vL-vC)/-B + 1/2*-B*((vL-vC)/-B)^2 - (vL * ((vL-vC)/-B)))) & vC >= 0 }
      ++
      { pL' = vL, pC' = vC, vC' = aC &
        (vC>=vL -> (pL-pC >= vC*(vL-vC)/-B + 1/2*-B*((vL-vC)/-B)^2 - (vL * ((vL-vC)/-B)))) & vC >= 0 }
    }
  }*
]
/* Safety condition */
( pC <= pL )
```

How can we identify these over-constraining errors?

```
/* Initial conditions */
( A > 0 & B > 0 & vL >0 & pC < pL &
(vC>=vL -> (pL-pC > vC*(vL-vC)/-B + 1/2*-B*((vL-vC)/-B)^2 - (vL * ((vL-vC)/-B)))))
->
[
  {
    /* Control */
    {
      /* Safely assign acceleration or braking for the controlled car */
       {?true; aC := A;}
       {?vC=0;aC := 0;}
       ++{aC := -B;}
    }
    /* Continuous dynamics and event triggers */
    {
      { pL' = vL, pC' = vC, vC' = aC &
        (vC>=vL -> (pL-pC <= vC*(vL-vC)/-B + 1/2*-B*((vL-vC)/-B)^2 - (vL * ((vL-vC)/-B)))) & vC >= 0 }
      ++
      { pL' = vL, pC' = vC, vC' = aC &
        (vC>=vL -> (pL-pC >= vC*(vL-vC)/-B + 1/2*-B*((vL-vC)/-B)^2 - (vL * ((vL-vC)/-B)))) & vC >= 0 }
    }
  }*
]
/* Safety condition */
( pC <= pL )
```



Proof: ✔ All goals closed

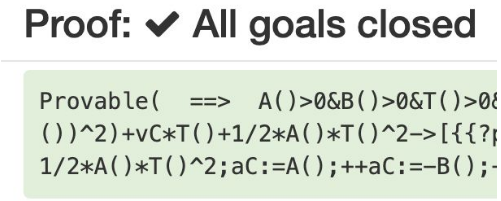
```
Provable(  ==>  A()>0&B()>0&T()>0
())^2)+vC*T()+1/2*A()*T()^2->[{{?
1/2*A()*T()^2;aC:=A();++aC:=-B();
```

How can we identify these over-constraining errors?

```
/* Initial conditions */
( A > 0 & B > 0 & vL >0 & pC < pL &
(vC>=vL -> (pL-pC > vC*(vL-vC)/-B + 1/2*-B*((vL-vC)/-B)^2 - (vL * ((vL-vC)/-B)))))
->
[
  {
    /* Control */
    {
      /* Safely assign acceleration or braking for the controlled car */
       {?(vC>=vL -> (pL-pC > vC*(vL-vC)/-B + 1/2*-B*((vL-vC)/-B)^2 - (vL * ((vL-vC)/-B)))); aC := A;}
       ++{?vC=0;aC := 0;}
       ++{aC := -B;}
    }
    /* Continuous dynamics and event triggers */
    {
      { pL' = vL, pC' = vC, vC' = aC &
        (vC>=vL -> (pL-pC <= vC*(vL-vC)/-B + 1/2*-B*((vL-vC)/-B)^2 - (vL * ((vL-vC)/-B))) & vC >= 0 }
      ++
      { pL' = vL, pC' = vC, vC' = aC &
        (vC>=vL -> (pL-pC >= vC*(vL-vC)/-B + 1/2*-B*((vL-vC)/-B)^2 - (vL * ((vL-vC)/-B))) & vC >= 0 }
    }
  }*
]
/* Safety condition */
( pC <= pL )
```

How can we identify these over-constraining errors?

```
/* Initial conditions */
( A > 0 & B > 0 & vL >0 & pC < pL &
 true )
->
[
 {
  /* Control */
  {
   /* Safely assign acceleration or braking for the controlled car */
    {?true; aC := A;}
    ++{?true;aC := 0;}
    ++{aC := -B;}
  }
  /* Continuous dynamics and event triggers */
  {
   { pL' = vL, pC' = vC, vC' = aC &
    (vC>=vL -> (pL-pC <= vC*(vC-vL)/-B + 1/2*-B*((vC-vL)/-B)^2 - (vL * ((vC-vL)/-B)))) & vC >= 0 }
   ++
   { pL' = vL, pC' = vC, vC' = aC &
    (vC>=vL -> (pL-pC >= vC*(vC-vL)/-B + 1/2*-B*((vC-vL)/-B)^2 - (vL * ((vC-vL)/-B)))) & vC >= 0 }
  }
 }*
]
/* Safety condition */
( pC <= pL )
```



Proof: ✓ All goals closed

```
Provable(  ==>  A()>0&B()>0&T()>0
())^2)+vC*T()+1/2*A()*T()^2->[{{?
1/2*A()*T()^2;aC:=A();++aC:=-B();
```

Overview of Implementation

Proof tree analysis

- Witnessed facts?
- Used facts?
- Unused facts?

New tactics

- minQE
- minAuto
- minAutoXtreme

Minimizing sequents: Guess & Check

Starting sequent: `⊢ x>0&y>0->[?x>=1;++?x<1;{x'=1&y>0}]x>0`

Candidate mutations:

Minimizing sequents: Guess

Starting sequent: `⊢ x>0&y>0->[?x>=1;++?x<1;{x'=1&y>0}]x>0`

Candidate mutations:

`⊢ true ->[?x>=1;++?x<1 ;{x'=1&y>0 }]x>0`

Minimizing sequents: Guess

Starting sequent: **`⊢ x>0&y>0->[?x>=1;++?x<1;{x'=1&y>0}]x>0`**

Candidate mutations:

⊢ **`true`** `->[?x>=1;++?x<1 ;{x'=1&y>0 }]x>0`

⊢ `x>0&`**`true`** `->[?x>=1;++?x<1 ;{x'=1&y>0 }]x>0`

Minimizing sequents: Guess

Starting sequent: `⊢ x>0&y>0->[?x>=1;++?x<1;{x'=1&y>0}]x>0`

Candidate mutations:

```
⊢  true       ->[?x>=1;++?x<1 ;{x'=1&y>0 }]x>0
⊢  x>0&true   ->[?x>=1;++?x<1 ;{x'=1&y>0 }]x>0
⊢  x>0&y>0    ->[?x>=1;++?x<1 ;{x'=1&true}]x>0
```

Minimizing sequents: Guess

Starting sequent: `⊢ x>0&y>0->[?x>=1;++?x<1;{x'=1&y>0}]x>0`

Candidate mutations:

⊢ **true** ->[?x>=1;++?x<1;{x'=1&y>0}]x>0

⊢ x>0&**true** ->[?x>=1;++?x<1;{x'=1&y>0}]x>0

⊢ x>0&y>0 ->[?x>=1;++?x<1;{x'=1&**true**}]x>0

⊢ x>0&y>0 ->[?x>=1;++?**true**;{x'=1&y>0}]x>0

Minimizing sequents: Guess

Starting sequent: `⊢ x>0&y>0->[?x>=1;++?x<1;{x'=1&y>0}]x>0`

Candidate mutations:

⊢ **true** ->[?x>=1;++?x<1;{x'=1&y>0}]x>0

⊢ x>0&**true** ->[?x>=1;++?x<1;{x'=1&y>0}]x>0

⊢ x>0&y>0 ->[?x>=1;++?x<1;{x'=1&**true**}]x>0

⊢ x>0&y>0 ->[?x>=1;++?**true**;{x'=1&y>0}]x>0

⊢ x>0&**true** ->[?**true**;++?**true**;{x'=1&**true**}]x>0

Minimizing sequents: Guess & **Check**

Starting sequent: **⊢ x>0&y>0->[?x>=1;++?x<1;{x'=1&y>0}]x>0**

Candidate mutations:

~~⊢ **true** ->[?x>=1;++?x<1 ;{x'=1&y>0 }]x>0~~

⊢ x>0&**true** ->[?x>=1;++?x<1 ;{x'=1&y>0 }]x>0

⊢ x>0&y>0 ->[?x>=1;++?x<1 ;{x'=1&**true**}]x>0

⊢ x>0&y>0 ->[?x>=1;++?**true**;{x'=1&y>0 }]x>0

⊢ x>0&**true** ->[?**true**;++?**true**;{x'=1&**true**}]x>0

Example/Demo

Overview of Implementation

New tactics

- minQE
- minAuto
- minAutoXtreme

Overview of Implementation

New tactics

Minimizing sequents: Guess & **Check**

Starting sequent: `⊢ x>0&y>0->[?x>=1;++?x<1;{x'=1&y>0}]x>0`

Candidate mutations:

~~`⊢ true ->[?x>=1;++?x<1 ;{x'=1&y>0 }]x>0`~~

`⊢ x>0&true ->[?x>=1;++?x<1 ;{x'=1&y>0 }]x>0`

`⊢ x>0&y>0 ->[?x>=1;++?x<1 ;{x'=1&true}]x>0`

`⊢ x>0&y>0 ->[?x>=1;++?true;{x'=1&y>0 }]x>0`

`⊢ x>0&true ->[?true;++?true;{x'=1&true}]x>0`

Overview of Implementation

Proof tree analysis

- Witnessed facts?
- Used facts?
- Unused facts?

New tactics

Minimizing sequents: Guess & **Check**

Starting sequent: **`⊢ x>0&y>0->[?x>=1;++?x<1;{x'=1&y>0}]x>0`**

Candidate mutations:

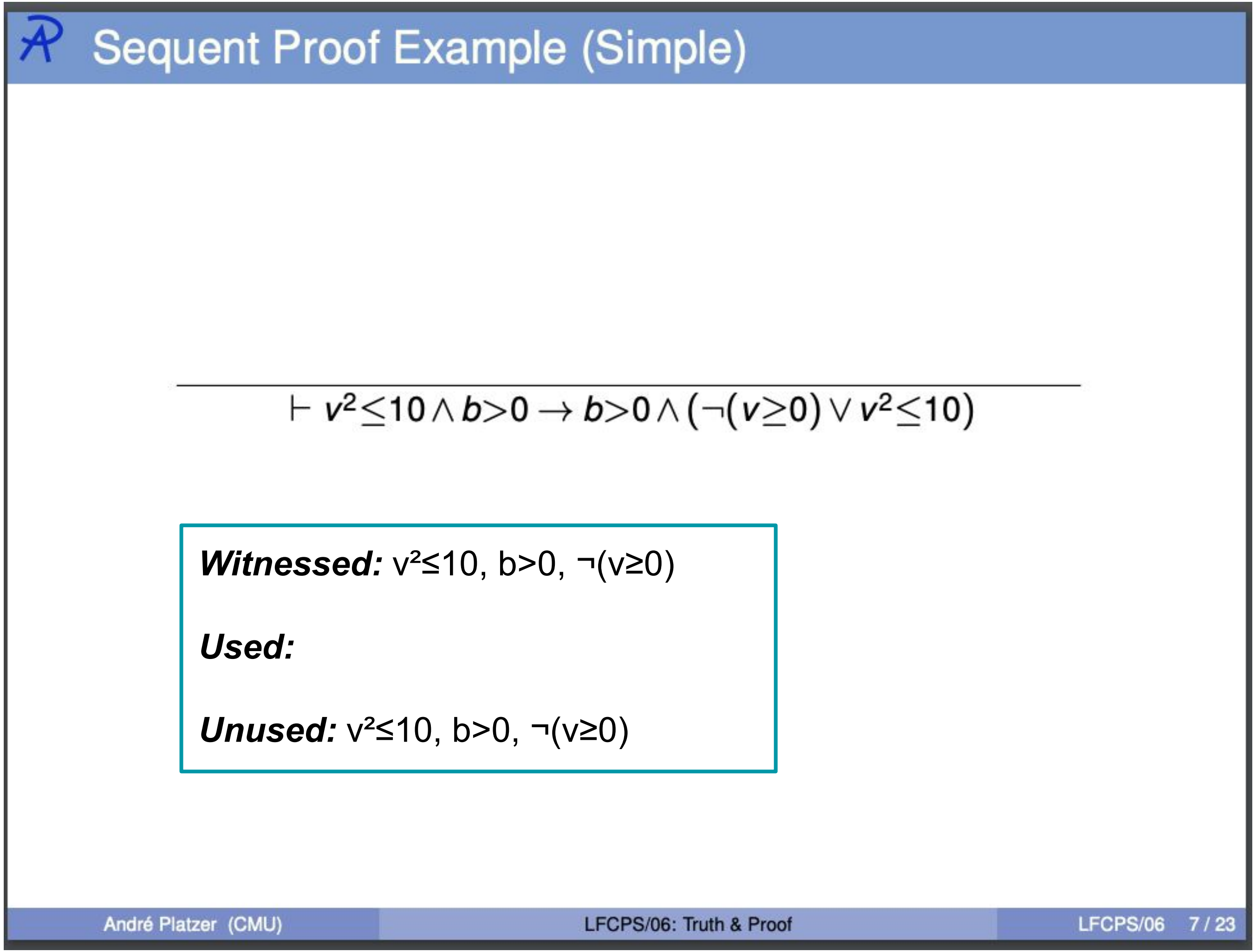
~~`⊢ true ->[?x>=1;++?x<1 ;{x'=1&y>0 }]x>0`~~

`⊢ x>0&true ->[?x>=1;++?x<1 ;{x'=1&y>0 }]x>0`

`⊢ x>0&y>0 ->[?x>=1;++?x<1 ;{x'=1&true}]x>0`

`⊢ x>0&y>0 ->[?x>=1;++?true;{x'=1&y>0 }]x>0`

`⊢ x>0&true ->[?true;++?true;{x'=1&true}]x>0`



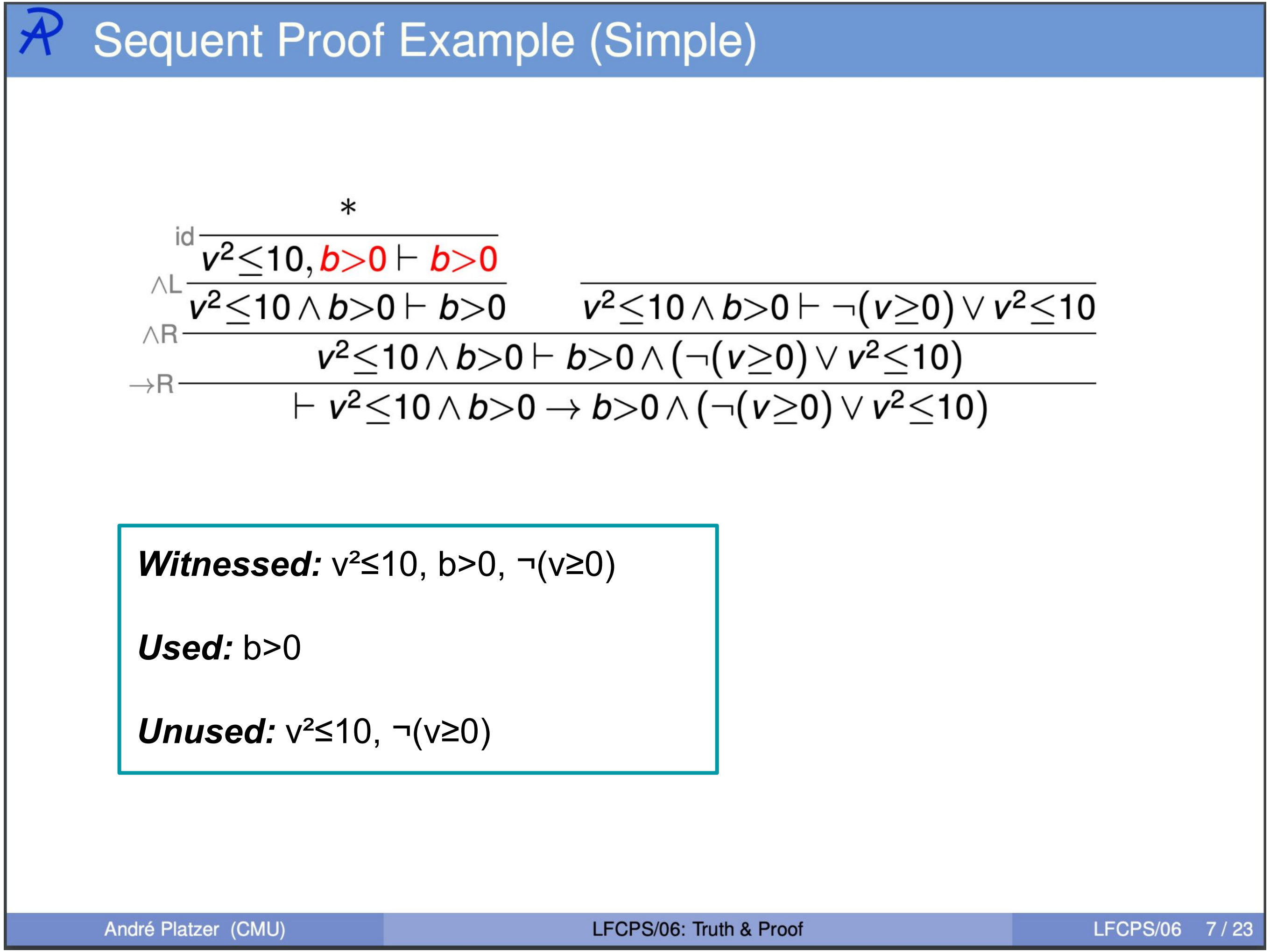
Sequent Proof Example (Simple)

$$\frac{}{\vdash v^2 \le 10 \land b>0 \to b>0 \land (\lnot(v \ge 0) \lor v^2 \le 10)}$$

Witnessed: $v^2 \le 10$, $b>0$, $\lnot(v \ge 0)$

Used:

Unused: $v^2 \le 10$, $b>0$, $\lnot(v \ge 0)$



Sequent Proof Example (Simple)

$$\to\!R\,\dfrac{\land R\,\dfrac{\land L\,\dfrac{\mathrm{id}\,\dfrac{*}{v^2\le 10, {\color{red}b>0} \vdash {\color{red}b>0}}}{v^2\le 10 \land b>0 \vdash b>0} \qquad \dfrac{}{v^2\le 10 \land b>0 \vdash \neg(v\ge 0) \lor v^2\le 10}}{v^2\le 10 \land b>0 \vdash b>0 \land (\neg(v\ge 0) \lor v^2\le 10)}}{\vdash v^2\le 10 \land b>0 \to b>0 \land (\neg(v\ge 0) \lor v^2\le 10)}$$

Witnessed: $v^2 \le 10$, $b>0$, $\neg(v\ge 0)$

Used: $b>0$

Unused: $v^2 \le 10$, $\neg(v\ge 0)$

Sequent Proof Example (Simple)

$$\to\!R\,\dfrac{\land R\,\dfrac{\land L\,\dfrac{\text{id}\,\dfrac{*}{v^2\le 10, b>0 \vdash b>0}}{v^2\le 10\land b>0 \vdash b>0} \qquad \lor R\,\dfrac{\land L\,\dfrac{\text{id}\,\dfrac{*}{{\color{red}v^2\le 10}, b>0 \vdash \neg(v\ge 0), {\color{red}v^2\le 10}}}{v^2\le 10\land b>0 \vdash \neg(v\ge 0), v^2\le 10}}{v^2\le 10\land b>0 \vdash \neg(v\ge 0)\lor v^2\le 10}}{v^2\le 10\land b>0 \vdash b>0\land(\neg(v\ge 0)\lor v^2\le 10)}}{\vdash v^2\le 10\land b>0 \to b>0\land(\neg(v\ge 0)\lor v^2\le 10)}$$

Witnessed: $v^2\le 10$, $b>0$, $\neg(v\ge 0)$

Used: $b>0$, $v^2\le 10$

Unused: $\neg(v\ge 0)$

Sequent Proof Example (Simple)

$$
{\scriptstyle \to R}\dfrac{{\scriptstyle \wedge R}\dfrac{{\scriptstyle \wedge L}\dfrac{{\scriptstyle \mathrm{id}}\dfrac{*}{v^2\leq 10, b>0 \vdash b>0}}{v^2\leq 10 \wedge b>0 \vdash b>0} \qquad {\scriptstyle \vee R}\dfrac{{\scriptstyle \wedge L}\dfrac{{\scriptstyle \mathrm{id}}\dfrac{*}{v^2\leq 10, b>0 \vdash \xcancel{\neg(v\geq 0)}, v^2\leq 10}}{v^2\leq 10 \wedge b>0 \vdash \xcancel{\neg(v\geq 0)}, v^2\leq 10}}{v^2\leq 10 \wedge b>0 \vdash \xcancel{\neg(v\geq 0)} \vee v^2\leq 10}}{v^2\leq 10 \wedge b>0 \vdash b>0 \wedge (\xcancel{\neg(v\geq 0)} \vee v^2\leq 10)}}{\vdash v^2\leq 10 \wedge b>0 \to b>0 \wedge (\xcancel{\neg(v\geq 0)} \vee v^2\leq 10)}
$$

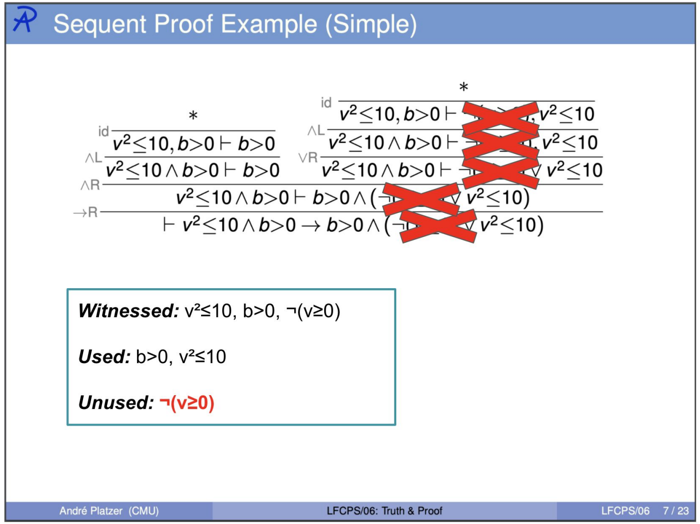
Witnessed: $v^2\leq 10$, $b>0$, $\neg(v\geq 0)$

Used: $b>0$, $v^2\leq 10$

Unused: **$\neg(v\geq 0)$**

Overview of Implementation

Proof tree analysis



Sequent Proof Example (Simple)

Witnessed: v²≤10, b>0, ¬(v≥0)

Used: b>0, v²≤10

Unused: ¬(v≥0)

André Platzer (CMU) LFCPS/06: Truth & Proof LFCPS/06 7 / 23

New tactics

Minimizing sequents: Guess & **Check**

Starting sequent: `⊢ x>0&y>0->[?x>=1;++?x<1;{x'=1&y>0}]x>0`

Candidate mutations:

~~⊢ **true** ->[?x>=1;++?x<1 ;{x'=1&y>0 }]x>0~~

⊢ x>0&**true** ->[?x>=1;++?x<1 ;{x'=1&y>0 }]x>0

⊢ x>0&y>0 ->[?x>=1;++?x<1 ;{x'=1&**true**}]x>0

⊢ x>0&y>0 ->[?x>=1;++?**true**;{x'=1&y>0 }]x>0

⊢ x>0&**true** ->[?**true**;++?**true**;{x'=1&**true**}]x>0

Rules for Suggested Mutations

Suggested Mutation (m)	***How m strengthens the model***
$\textsc{Sm:AssumptionR}$ $\dfrac{A \vdash [\alpha]P}{m_1(A) \vdash [\alpha]P} \quad (a \in A)$ $m_1(\cdot) = \{\cdot \mapsto \cdot \setminus \{a\}\}$	● removes an assumption ● *m*-mutated dL formula is valid
$\textsc{Sm:AssumptionW}$ $\dfrac{A \vdash [\alpha]P}{m_2(A) \vdash [\alpha]P} \quad (p(a_1, a_2) \in A)$ $m_2(\cdot) = \{p(a_1, a_2) \mapsto p'(a_1, a_2)\}$	● weakens an assumption by replacing a1 < a2 with a1 ≤ a2 ● *m*-mutated dL formula is valid
$\textsc{Sm:PostA}$ $\dfrac{A \vdash [\alpha]P}{A \vdash [\alpha]m_7(P)} \quad (a \notin const(A), a \in A, a \notin P)$ $m_7(\cdot) = \{P \mapsto P \cup \{a\}\}$	● adds a post-condition ● *m*-mutated dL formula is valid

Formal Guarantees

- ***Termination:***
    - The analysis terminates.
    - Proof sketch: well-ordering argument over the # of proof steps, # of mutable facts, and remaining test time per fact
- ***Soundness:***
    - m-mutated models…
        - are valid,
        - prove the same post-conditions with fewer facts, and
        - prove stronger post-conditions with the same facts.
    - Proof sketch: rule induction over suggested mutations

Contributions

- An analysis for diagnosing modeling errors
  - Implementation
    - https://github.com/gaffordb/KeYmaeraX-release
  - Examples
  - Candidate and suggested mutations
  - Formal guarantees
    - Soundness
    - Termination