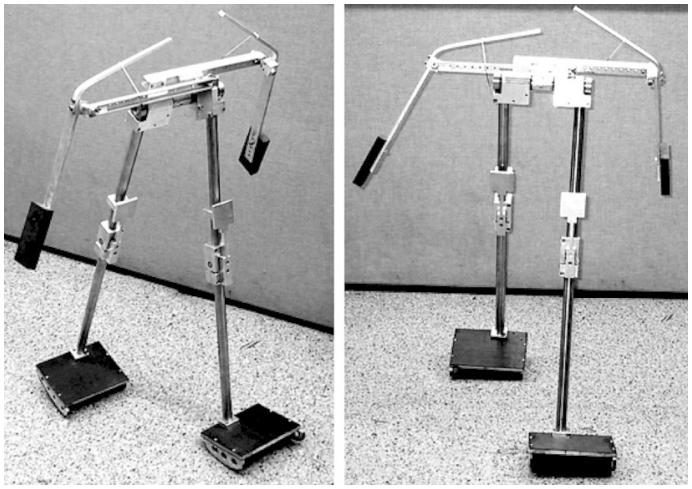
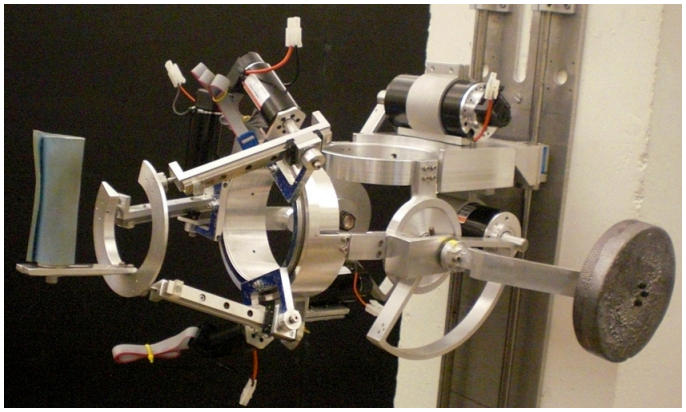
Enclosing Hybrid Behavior

Walid Taha, Halmstad University and Rice University

Joint work with
Michal Konecny (Aston), Jan Duracz (Halmstad), Adam Duracz (Halmstad),
and Aaron Ames (Texas A&M)

Model-based robot design

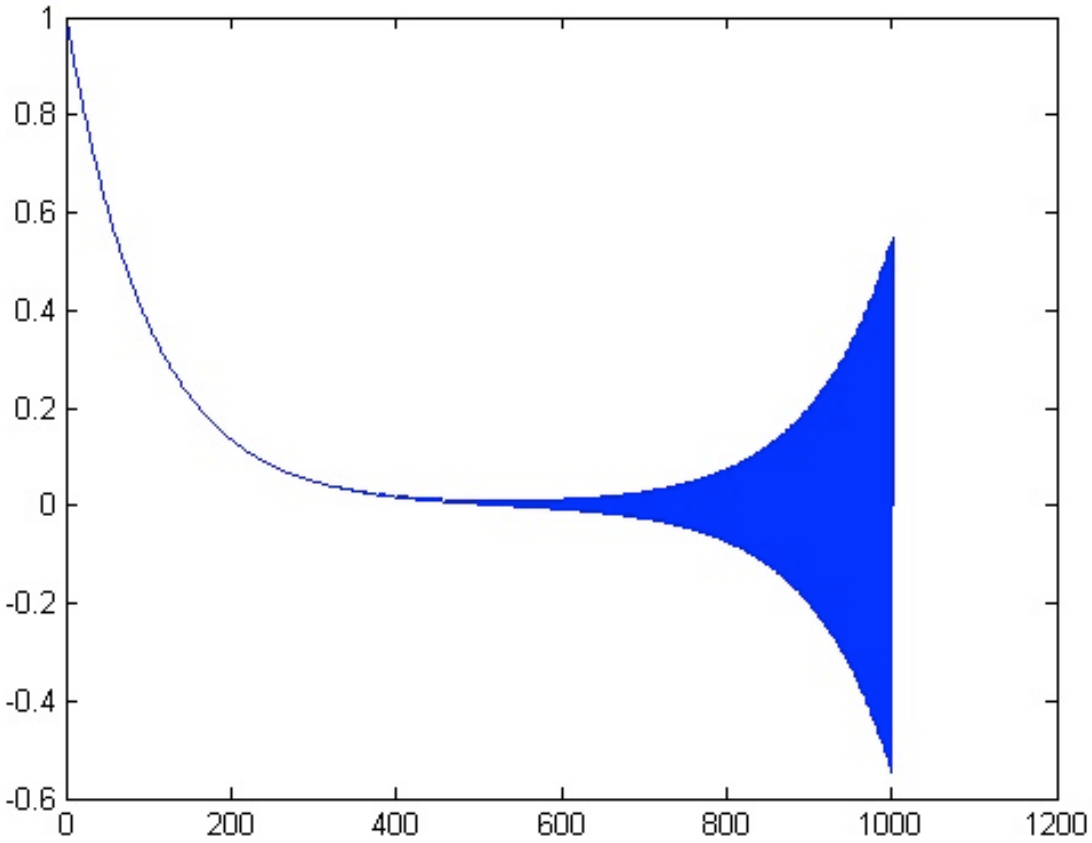


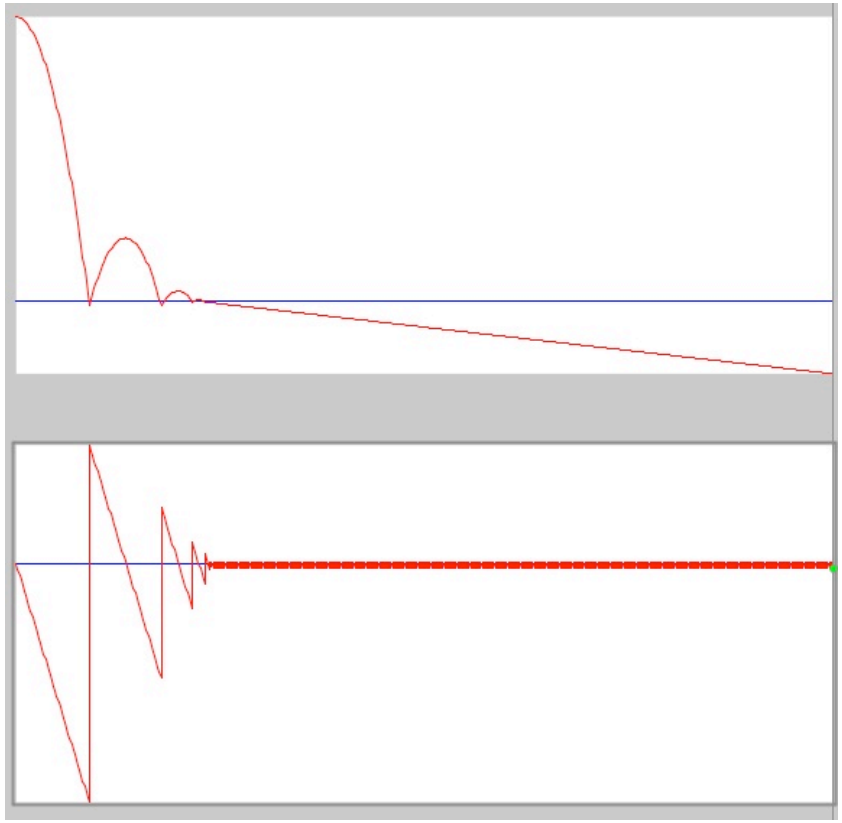


Simulation tools today

- No guarantee that behavior computed is consistent with model used.
  - Numerical artifacts
  - Integration drift
  - Singularities often ignored
  - Zeno behavior

Examples: Cont. and Hy.





Rest of this talk

- Enclosure methods
- Enclosing continuous behaviors
- Enclosing hybrid systems
  - Event detection and reset maps
  - Zeno behavior
- Conclusions

Idea: Enclosure methods

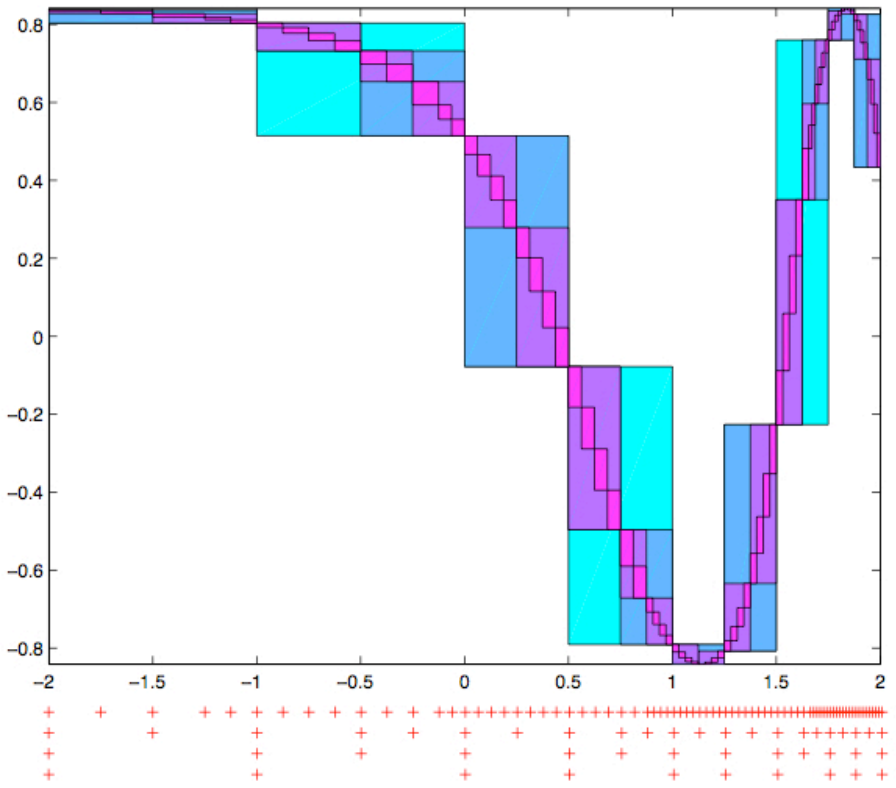


Figure 5.14: Refined enclosures of the integrand $f(x) = \sin(\cos(e^x))$.

- Always guarantee that solution is enclosed
- Can compute more precise answers as needed
- But can they be mechanized?

Continuous behaviors

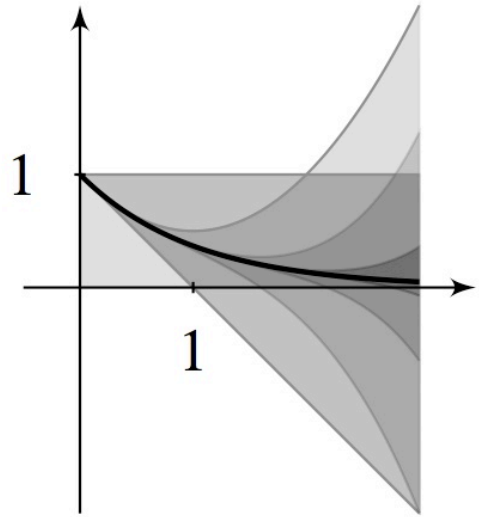
- An elegant, very general method exists:
  - Picard iteration
- Key challenge: Extending to proper enclosures

Set

$$\varphi_0(t) = y_0$$

and

$$\varphi_{k+1}(t) = y_0 + \int_{t_0}^{t} f(s, \varphi_k(s))\, ds.$$



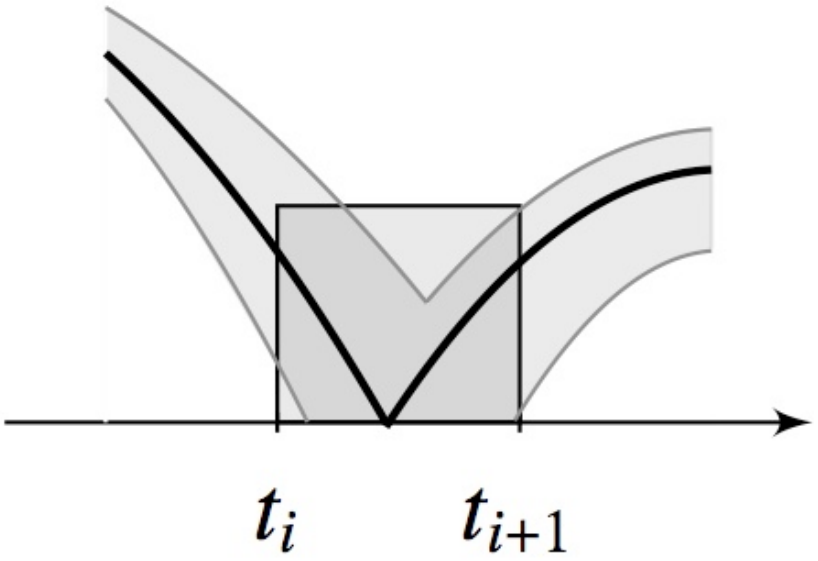
Example

```
class Main(simulator)
private x:=1; x':=0; x'':=0; mode := ""; end
  switch mode
    case ""
      x'' = -x
  end;
  simulator.endTime := 2.0;
  simulator.minSolverStep := 0.1;
end
```

Example

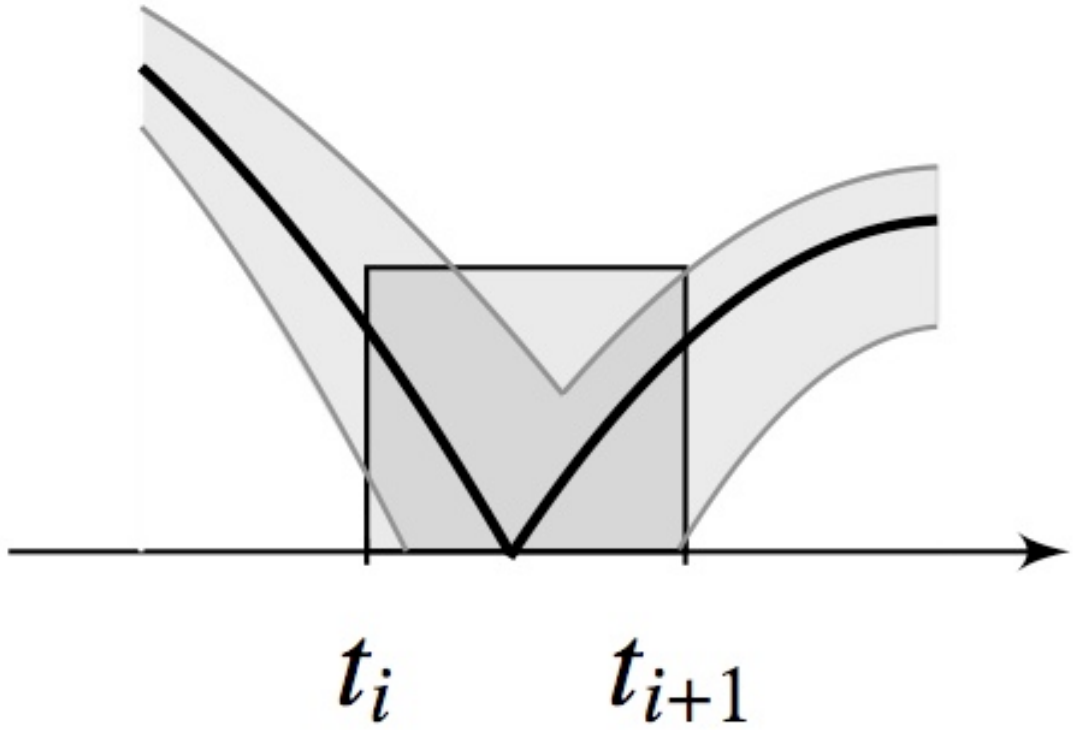
Event detection

- Enclosures provide a natural method for event detection (root find)
- Basic idea:
  - Mean value theorem
  - It's OK to say "I don't know"



Reset maps

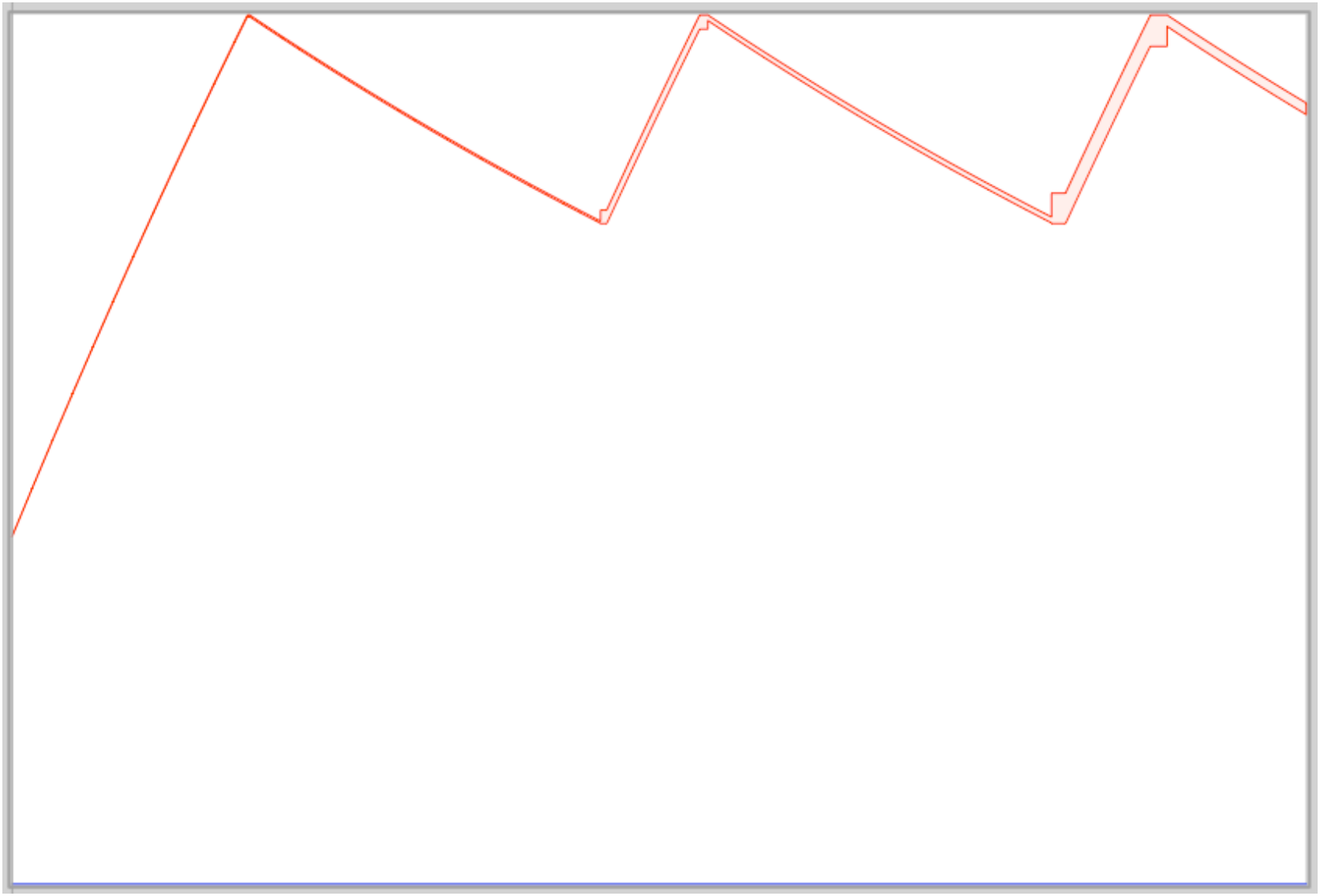
- Assume worst case behavior
- Note: Still need to know it was only *one* event that occurred in that interval



Example

```
class Main (simulator)
  private
    mode := "on"; x := 10; x' := 0;
  end
  switch mode
    case "on" require x <= 25
      if x == 25
        mode := "off"
      end;
      x' = 100 - x;
    case "off" require x >= 19
      if x == 19
        mode := "on"
      end;
      x' = - x;
  end;
  simulator.endTime := 1;
  simulator.minSolverStep := 0.01;
  simulator.minLocalizationStep := 0.001;
  simulator.minComputationImprovement := 0;
end
```

Example

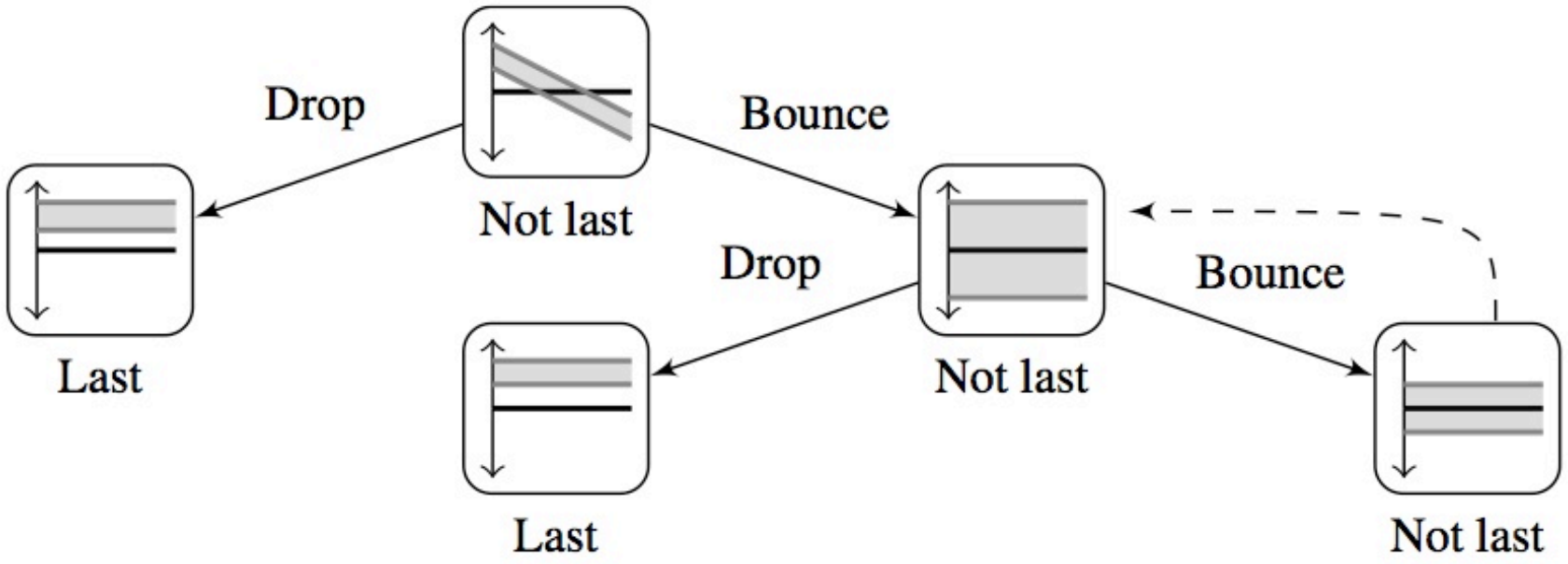


Zeno Behavior

- A bouncing ball comes to rest in finite time, with an infinite number of bounce events...
- A real problem for rigid body dynamics (with impacts)
- Directly relates to the issue of knowing that an event occurs exactly once in a given interval

Enclosing Zeno

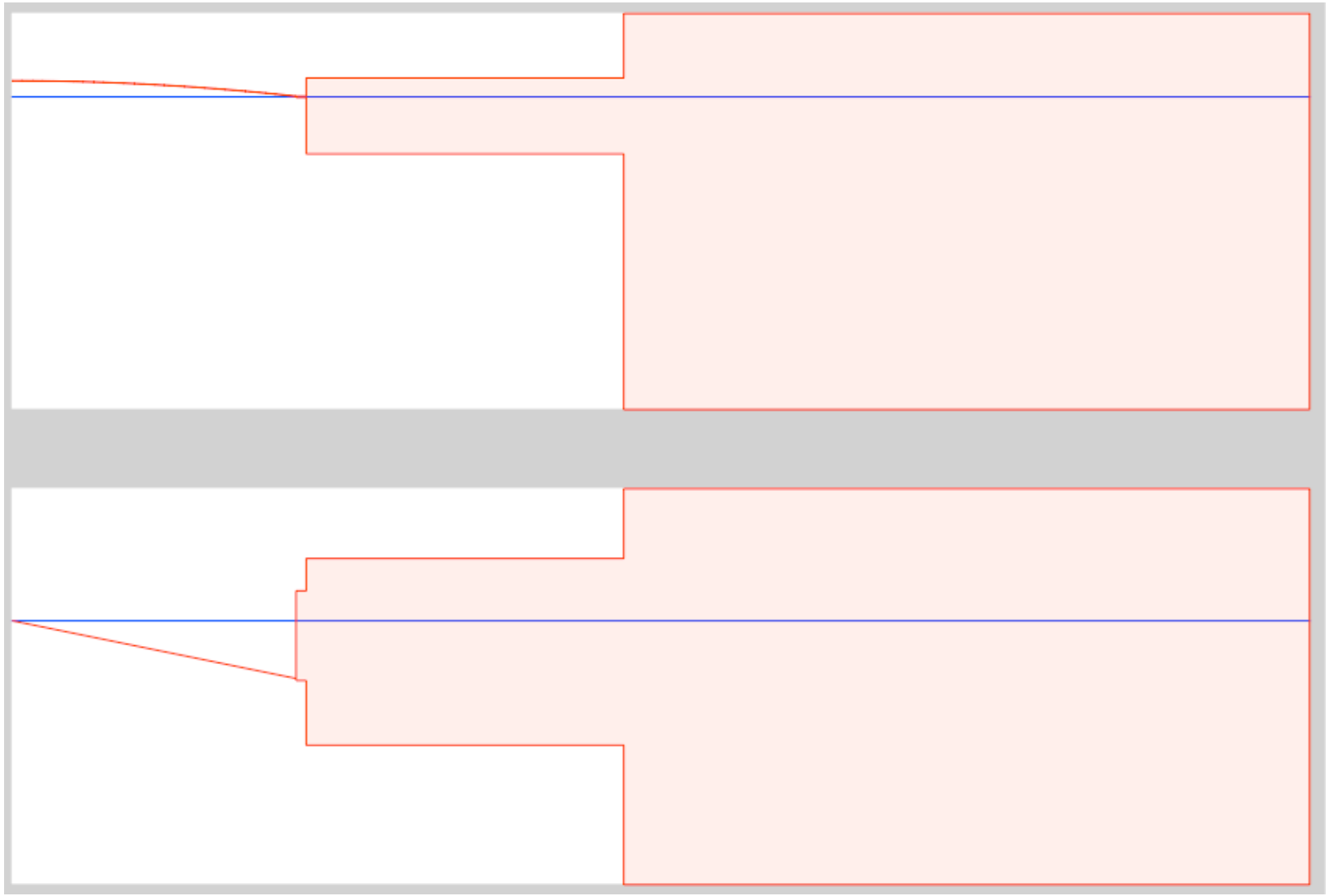
- Idea: We can actually relax that requirement if we know that a repeat event does NOT enlarge the enclosure we start with



Enclosing Zeno, Take I

```
class Main(simulator)
  private
    mode := "Fly";
    x    := 5;
    x'   := 0;
    x''  := 0;
  end
  switch mode
    case "Fly"
      if x == 0 && x' <= 0
        x' := -0.5*x';
        mode := "Fly";
      end;
      x'' = -10;
  end;
  simulator.endTime := 4.5;
  simulator.minSolverStep := 0.01;
  simulator.minLocalizationStep := 0.01;
  simulator.minComputationImprovement := 0.001;
end
```

Enclosing Zeno, Take I



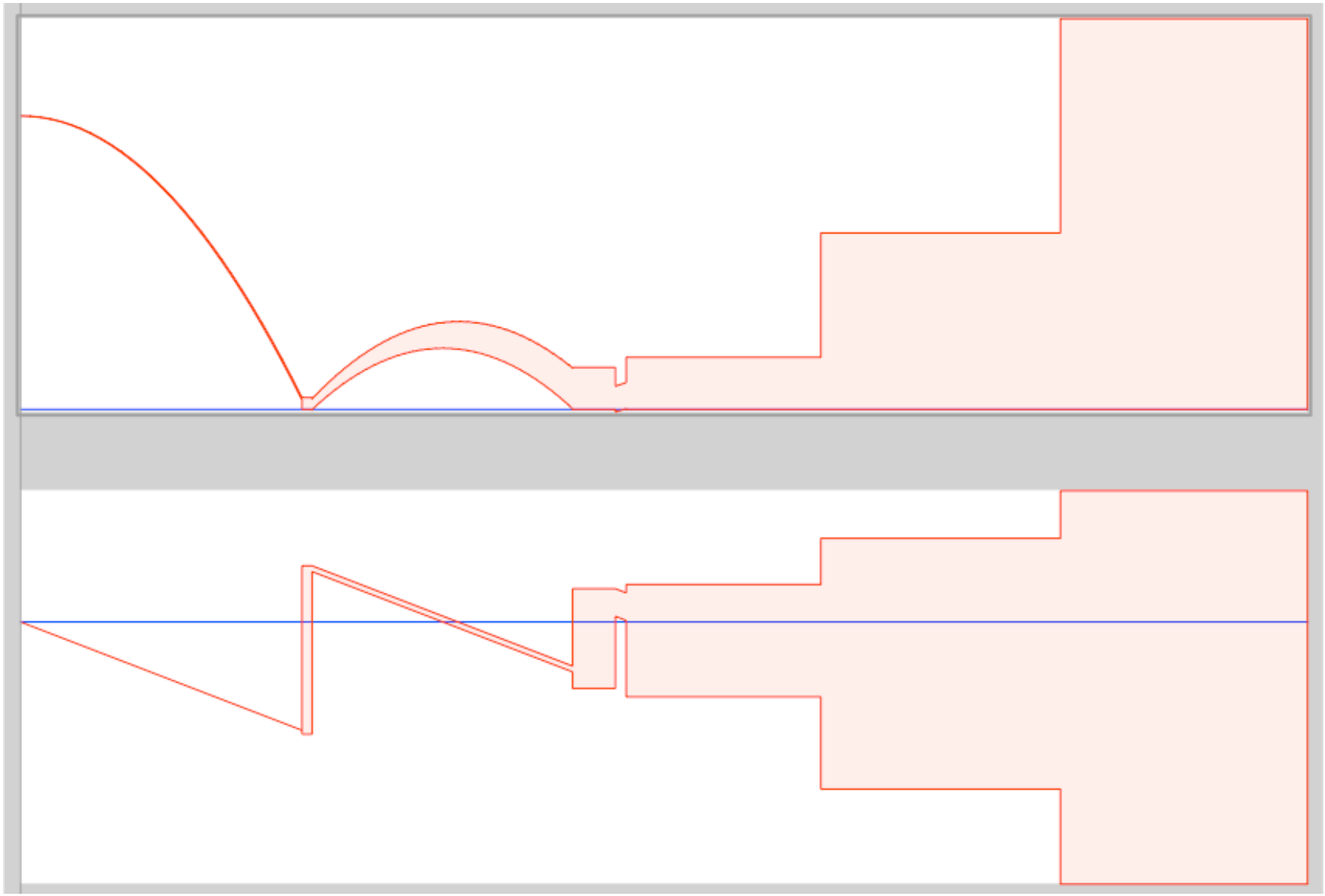
Fix: Over-constraining

- Enforce domain constraints (intersect)
  - Example: x >= 0
- Constraining speed based on explicit energy
  - Example: A notion of energy

Take II

```
class Main(simulator)
  private
    mode := "Fly";
    x    := 5;
    x'   := 0;
    x''  := 0;
  end
  switch mode
    case "Fly"
      require x >= 0 // New constraint
      if x == 0 && x' <= 0
        x' := -0.5*x';
        mode := "Fly";
      end;
      x'' = -10;
  end;
  simulator.endTime := 4.5;
  simulator.minSolverStep := 0.01;
  simulator.minLocalizationStep := 0.01;
  simulator.minComputationImprovement := 0.001;
end
```

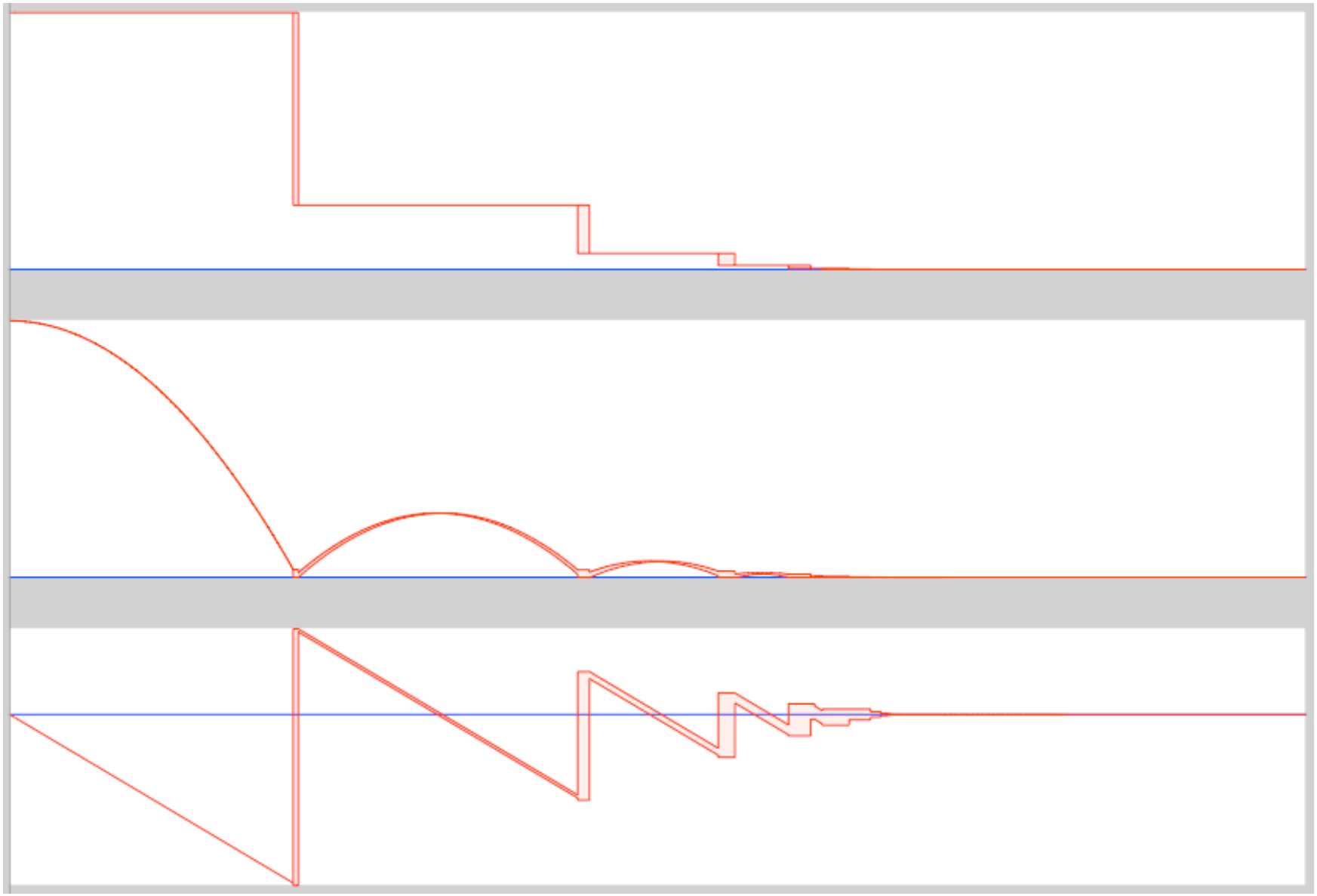
Take II



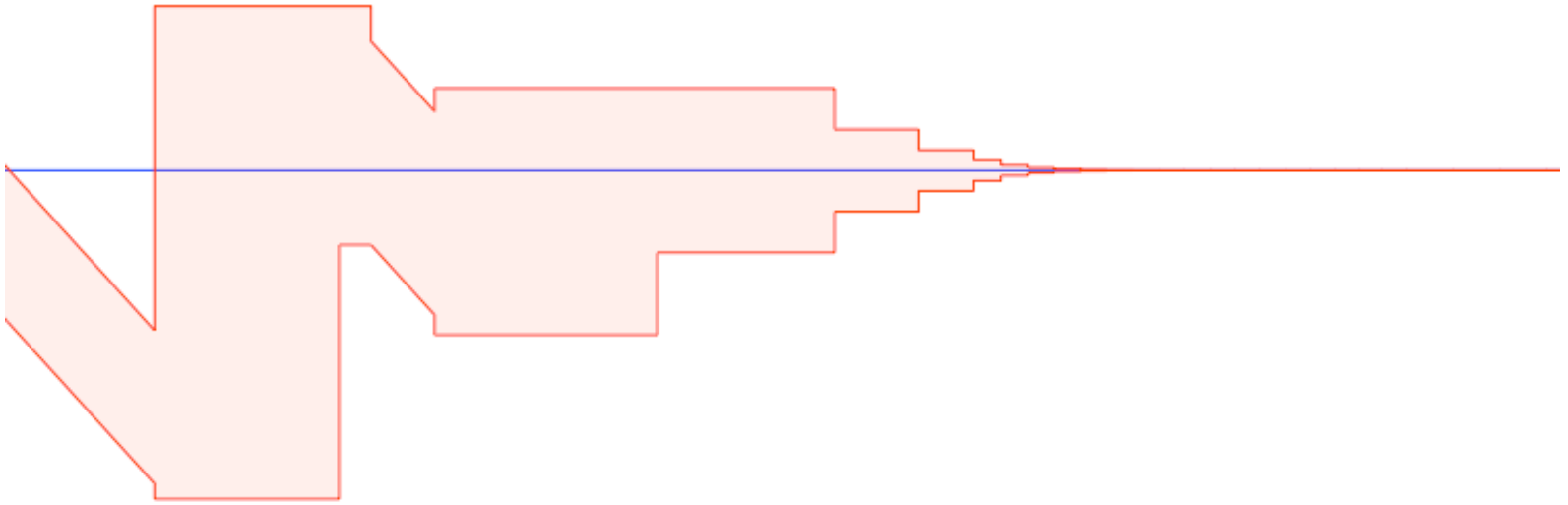
Take III

```
class Main(simulator)
  private
    mode := "Fly";
    x   := 5;
    x'  := 0;
    x'' := 0;
    r   := 100;  // Estimate of (twice) the energy
    r' := 0;
  end
  switch mode
    case "Fly"
      require x >= 0 &&
              r == x'*x' + 20*x //
      if x == 0 && x' <= 0
        x' := -0.5*x';
        r  := 0.25*r;     //
        mode := "Fly";
      end;
      x'' = -10;
      r'  = 0;
  end;
  simulator.endTime := 4.5;
  simulator.minSolverStep := 0.01;
  simulator.minLocalizationStep := 0.01;
  simulator.minComputationImprovement := 0.001;
end
```

Take III



Empire State Building



Technical results

- Proper interval Picard converges
- Event detection is sound
- Zeno method is sound

Conclusions

- Using enclosures
  - ensures that any answer produced is correct
  - simplifies correct event detection
  - admits an elegant way of handling certain classes of Zeno behavior
  - benefits from over-constraining

Future work

- Understanding algorithmic complexity
- Understanding performance on larger models (mainly drawn from the robotics domain)
- Identifying heuristics to limit loss of precision during continuous segments

Thank you!

- Checkout acumen-language.org