Keymaera Evaluator for Reliable and Robust Cyber-physical Hybrid-program Engineering via Novel Graphics

Justin Kerr (jgkerr) and Jasmine Cheng (jacheng)
Labs in this class

\[ P \rightarrow [H]Q \]

Hard parts:
1. design H (controller, system model)
2. prove Q holds (safety, correctness properties)
Hybrid Program Basics

P, Q are logical formulas

Hybrid program H is made of

- $x:=e$: Assign, set state variable $x$ to expression $e$
- $A;B$: Compose, $A$ then $B$
- $A \cup B$: Choice, $A$ or $B$
- $?A$: Test, if $A$ passes, the program can run
- $\{f'=y \& Q\}$: ODE, program follows $f'=y$ for some time while maintaining $Q$
- $A^*$: Loop, repeat $A$ for any number of iterations
Painful debugging

Differential drive controller from lab 4

```plaintext
{trackr:=*;?!trackr=0;cx:=x-dy*trackr;
 cy:=y+dx*trackr;
 !(cx-rogx)^2+(cy-rogy)^2>(buffer+abs(trackr))^2
 !(cx-rogx)^2+(cy-rogy)^2 < (buffer-abs(trackr))^2;
 a:=*;?a>=-B&a<=A};
 t:=0;
 \{v'=a,x'=dx*v,y'=dy*v,dx'=-v*dy/trackr,
  dy'=v*dx/trackr,t'=1&t<=T&v>=0
 }
}
```
Our project

- Debug hybrid programs through interactive execution
- Aimed at supporting course labs
Past Projects

Yu 2017
User makes all choices
Built on Keymaera X’s parser
User Flow

Annotate Model (text file) → Verify parse → Open visualizer

Automatic Playback

Manual branch selection
System Tour

Parser

Program Tree

ODE Solver

Simulator

Trace Selection

Auto

Manual

Visualization

States
Program tree

(((?canAccelerate; a:=A) U a:=B); t:=0; {ODE})*)
Program Traces

Given a start state,

\[ \text{trace} = (\text{choices list, end state}) \]

Why?

- Allows choice selection
- Prune invalid runs, prevent failure at runtime
- Natural tree-search structure
Program Trace Example

```
Program Trace Example
```

Diagram showing a program trace example with a tree structure and different branches representing choices and ODE run times.

Right side of the diagram displays a decision tree with choices and corresponding ODE run times:
- L, 0
- L, 0.1
- R, 0
- R, 0.1
- R, 10

The diagram also includes a section on a Simulator with options for Trace Selection (Auto, Manual) and States.
ODE and * trace generation

“\{x’=1\}” ← How long should we run an infinite ODE?
“x:=*” ← What value do we pick for a random assign?

ODE Branch based on execution time
* User annotates discretized range for branching
Manual Trace Selection

▷ Visualize all possible end states
▷ User selects one
Auto trace selection

“Distance” heuristic: pick traces close to post-condition truth boundaries since we always start inside a true region.
Auto trace selection

Truth distance $d$: formula $\rightarrow$ real

\[
d(e_1 \{<=, >=, <, >\} e_2) := |e_1 - e_2|
\]

\[
d(e_1 = e_2) \text{ undefined} \leftarrow \text{ignore equality because of numerical error}
\]

\[
d(P_1 \text{ and } P_2) := \min(d(P_1), d(P_2))
\]

\[
d(P_1 \text{ or } P_2) := \max(d(P_1), d(P_2))
\]
Visualization

- Use choices from trace
- Build up a list of states by computing ODE at time stamps
  - Runge-Kutta integration
- Draw robot states
Implementation

- Visualization: TKinter
- Minimal external libraries
- Parsing from scratch
Demos
Motivating labs

Lab 3: Robot on Racetracks

Lab 4: Differential Drive
Manual Mode: Lab 4 Demo
Catching Modelling Errors

A. swapped $x'$ and $y'$, B. missing negative sign, C. forgetting to scale by track radius, D. typo on $dx/dy$, E. swapped $dx/dy$, F. correct version
Catching Controller Errors

Controller deems these paths unsafe
Catching Controller Errors
Auto mode: faulty lab 4 results in collision
Auto mode: safe lab 4, risky trajectory for correct controller
Auto Mode: broken lab 3
Auto mode: safe lab 3
Questions?
Supplements
Future Work

1. Multi-level game tree search
2. Differential heuristic
3. User experience
4. Numerical Error
System Architecture

- Text input
- Parser (recursive)
  - Find top-level operator
  - Create syntax objects
- Simulator
  - Expand traces
  - Trace Selection
    - Auto: Choice heuristic on postcondition
    - Manual: Show outcomes; user chooses
- ODE Solver
- Evaluation program states
- States list
- Visualization
Handling loops

- Strip off top-level loop
- Run hybrid program inside the loop (manual or auto)
- Repeat

```
(((?canAccelerate; a:=A) ∪ a:=B); t:=0; {ODE})*
```

* Doesn’t allow for nested loops