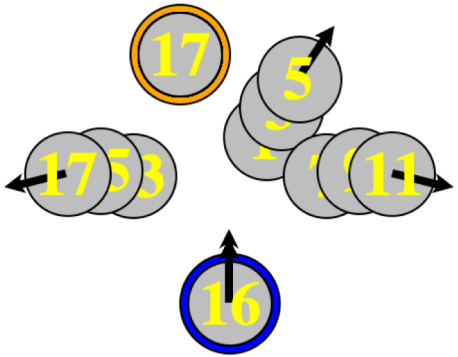
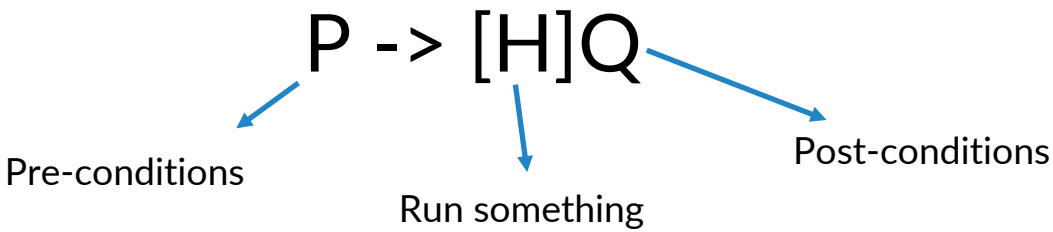
Keymaera Evaluator for Reliable and Robust Cyber-physical Hybrid-program Engineering via Novel Graphics

Justin Kerr (jgkerr) and Jasmine Cheng (jacheng)



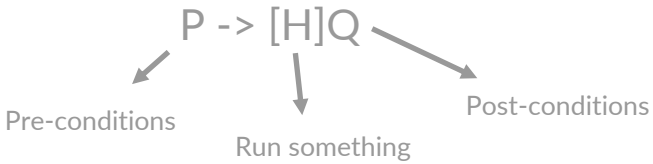
Labs in this class



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 ∪ 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

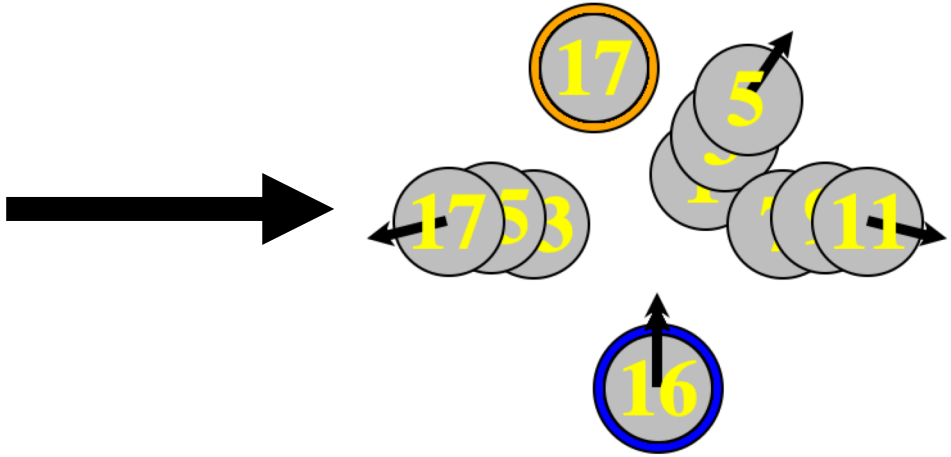
Differential drive controller from lab 4

```
{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

```
{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
  }
}*
```

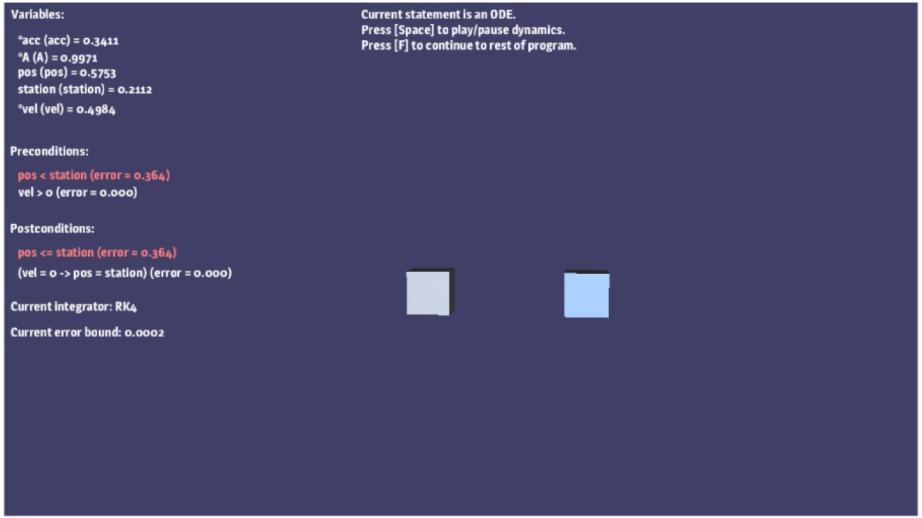


Past Projects

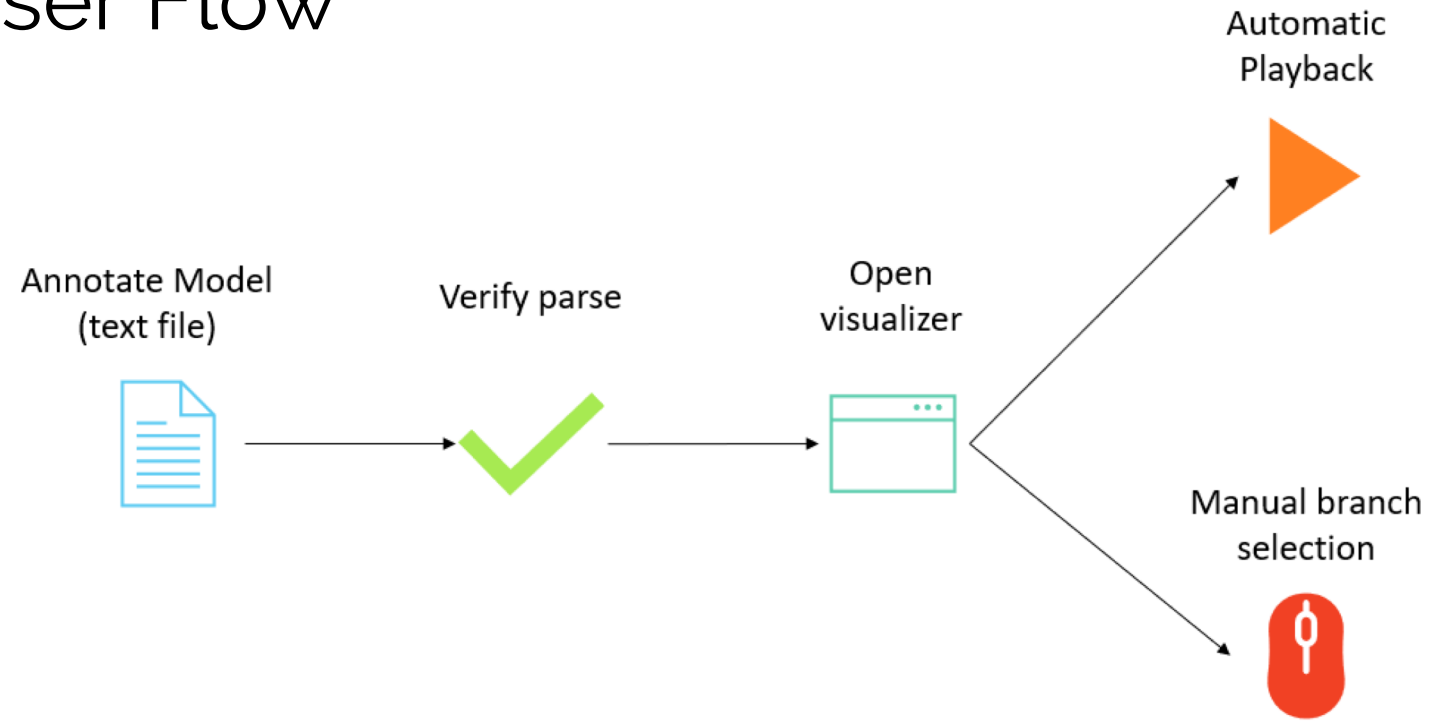
Yu 2017

User makes all choices

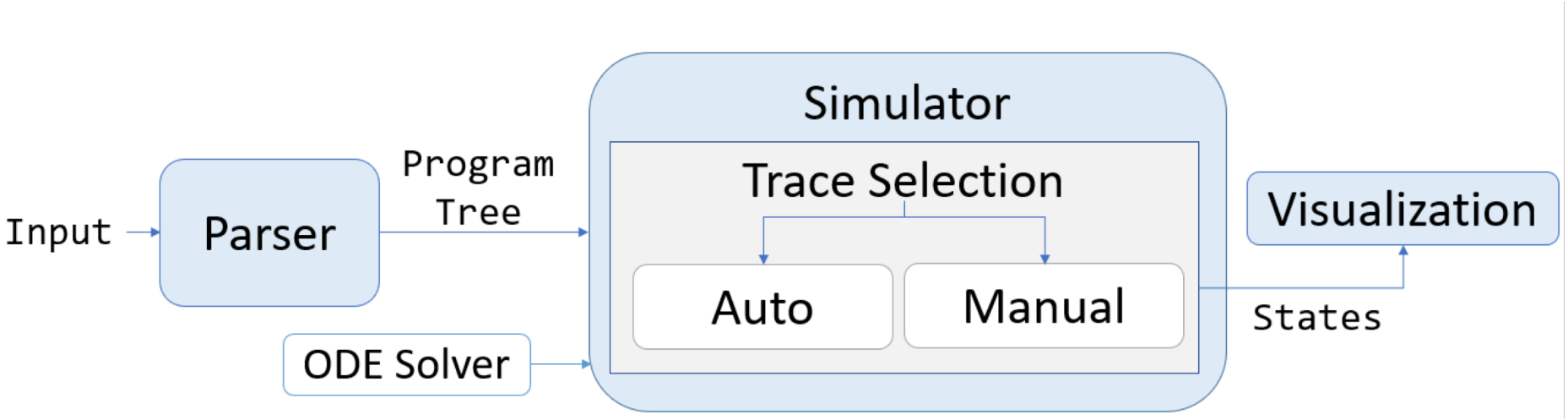
Built on Keymaera X's parser



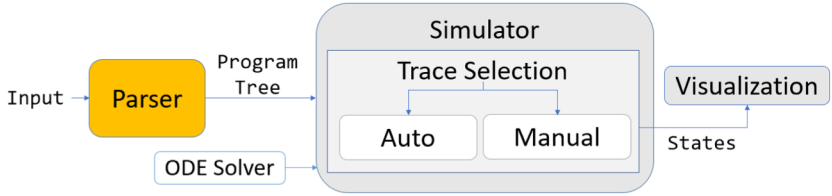
User Flow


Automatic
Playback
Annotate Model
(text file)
Verify parse
Open
visualizer
Manual branch
selection

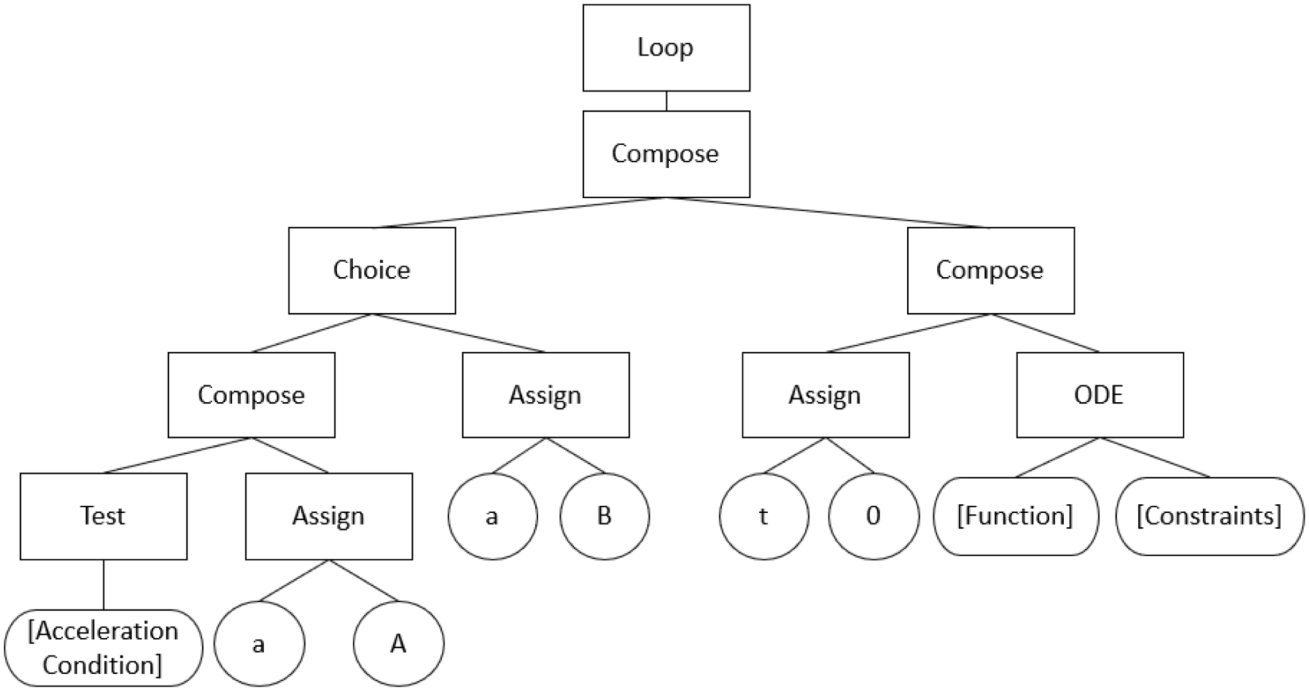
System Tour

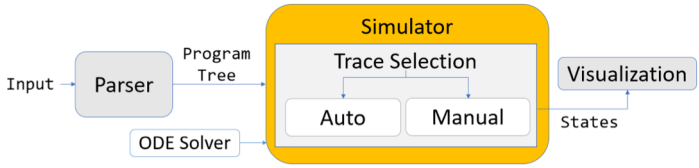

Input
Parser
Program
Tree
Simulator
Trace Selection
Auto
Manual
ODE Solver
Visualization
States

Program tree



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





Program Traces

Given a start state,

trace = (choices list, end state)

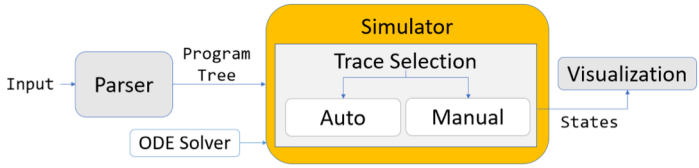
Why?

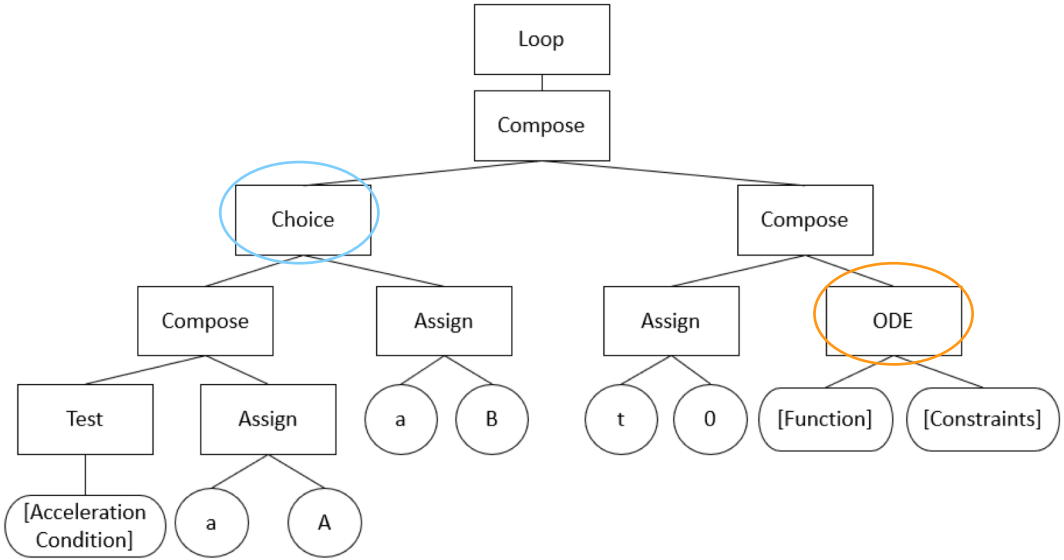
Allows choice selection

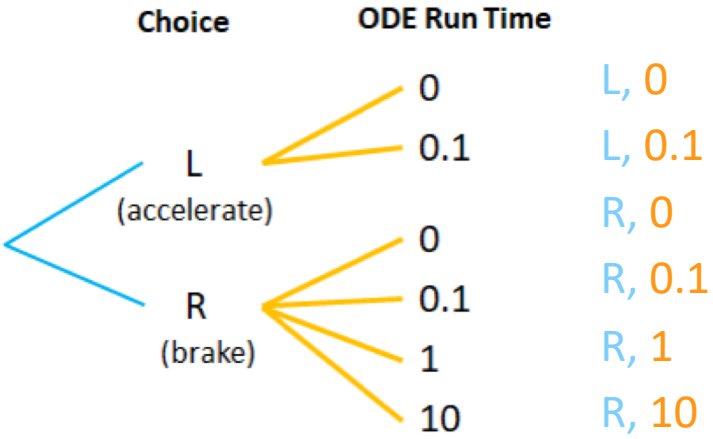
Prune invalid runs, prevent failure at runtime

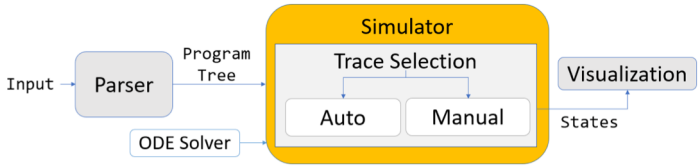
Natural tree-search structure

Program Trace Example









ODE and * trace generation

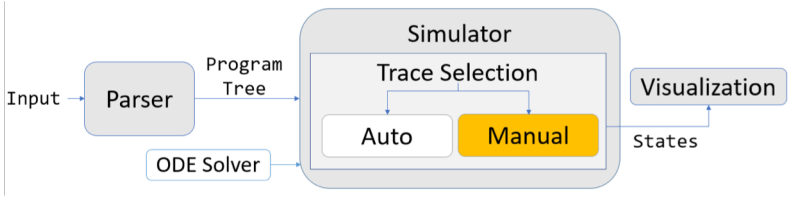
“{x’=1}” ← How long should we run an infinite ODE?

“x:=*” ← What value do we pick for a random assign?

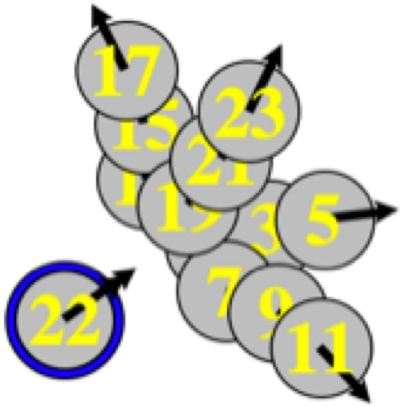
ODE Branch based on execution time

* User annotates discretized range for branching

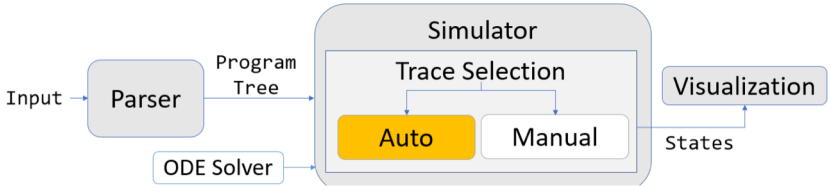
Manual Trace Selection



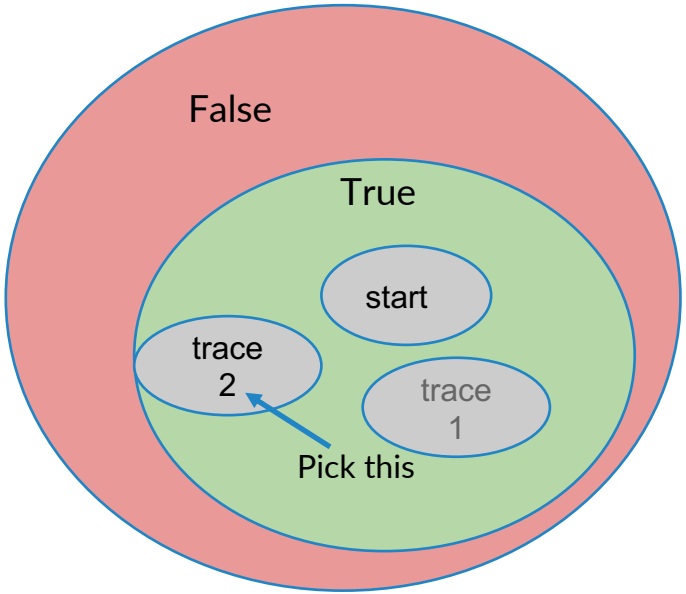
- Visualize all possible end states
- User selects one



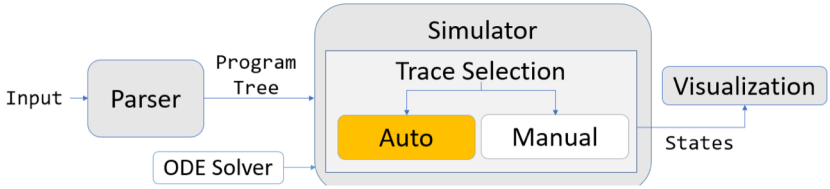
Auto trace selection



“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 → real

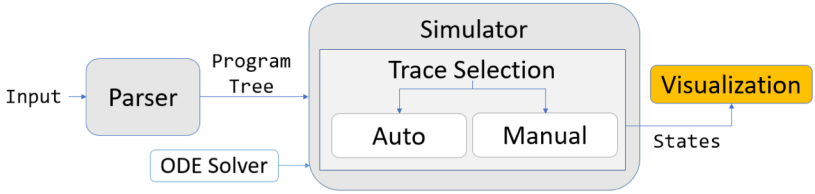
d(e_1 {<=, >=, <, >} e_2) := |e1-e2|

d(e_1 = e_2) undefined ← ignore equality because of numerical error

d(P_1 and P_2) := min(d(P_1), d(P_2))

d(P_1 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

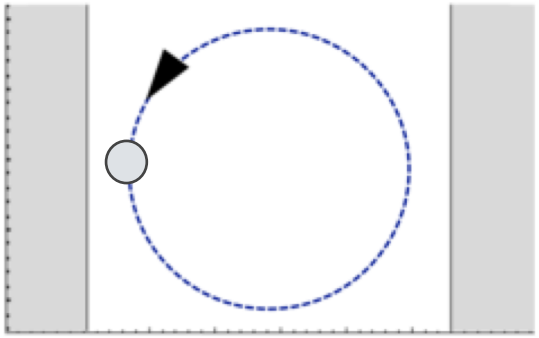
Implementation

- Visualization: TKinter
- Minimal external libraries
- Parsing from scratch

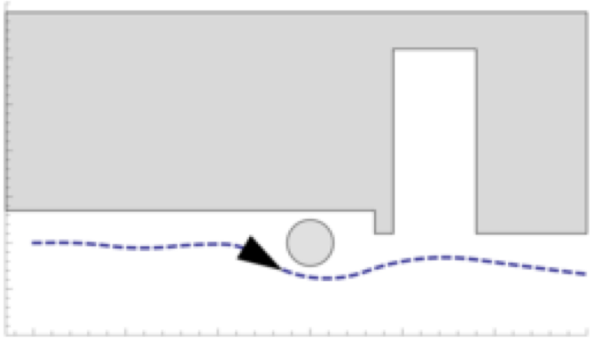
Demos

Motivating labs

Lab 3: Robot on Racetracks

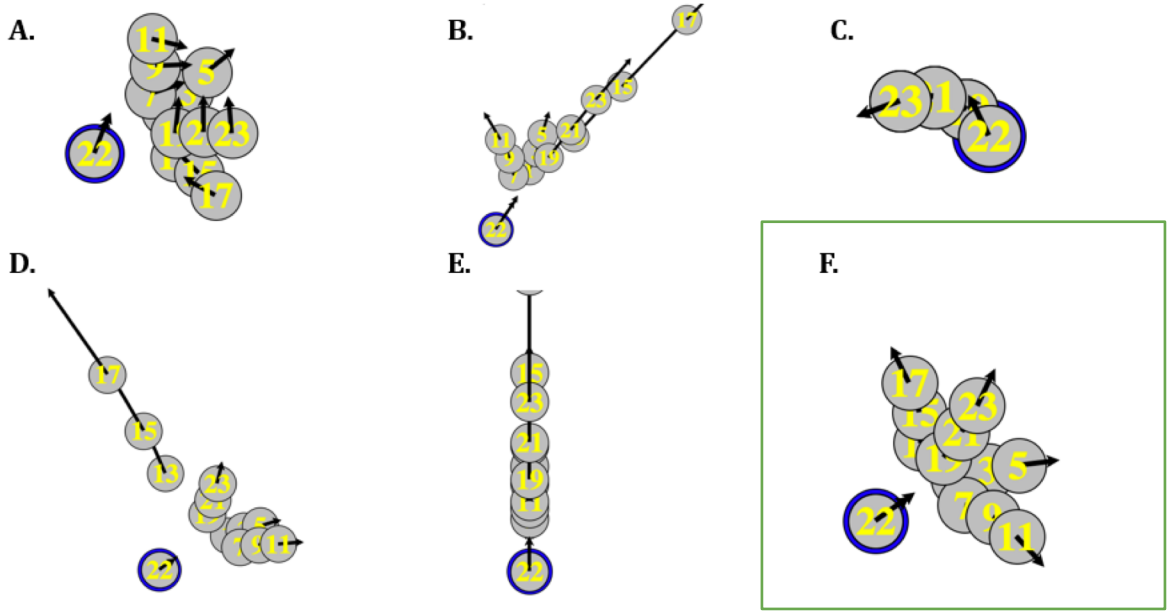


Lab 4: Differential Drive



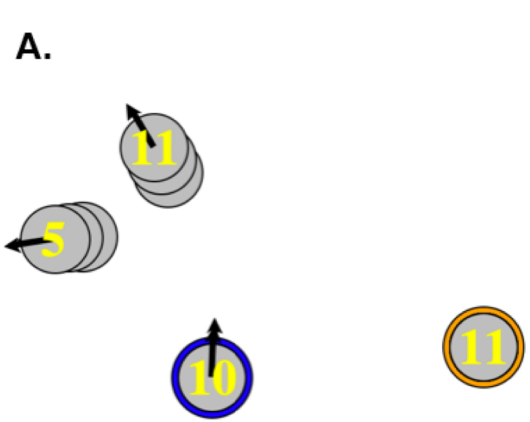
Manual Mode: Lab 4 Demo

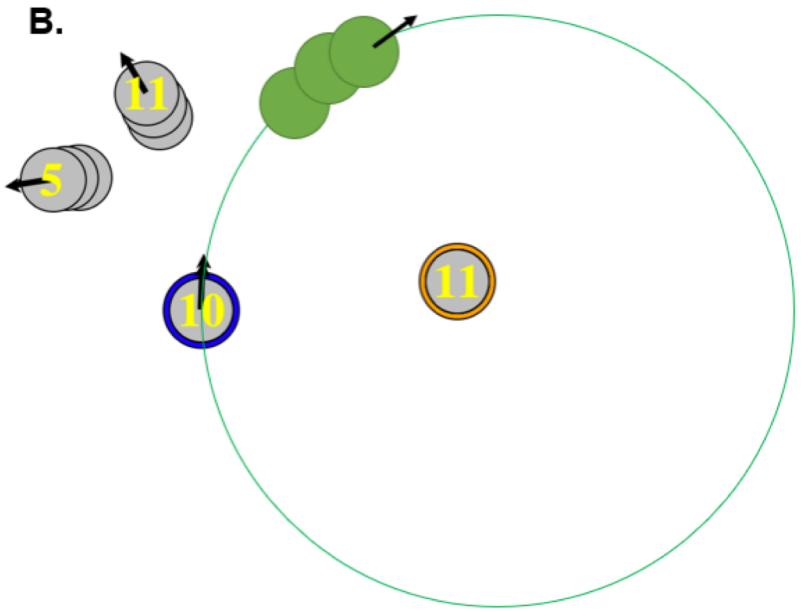
Catching Modelling Errors



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

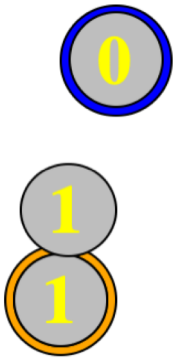
Catching Controller Errors





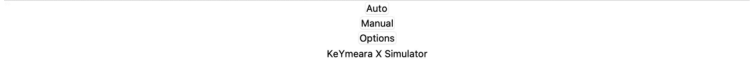
Controller deems these paths unsafe

Catching Controller Errors



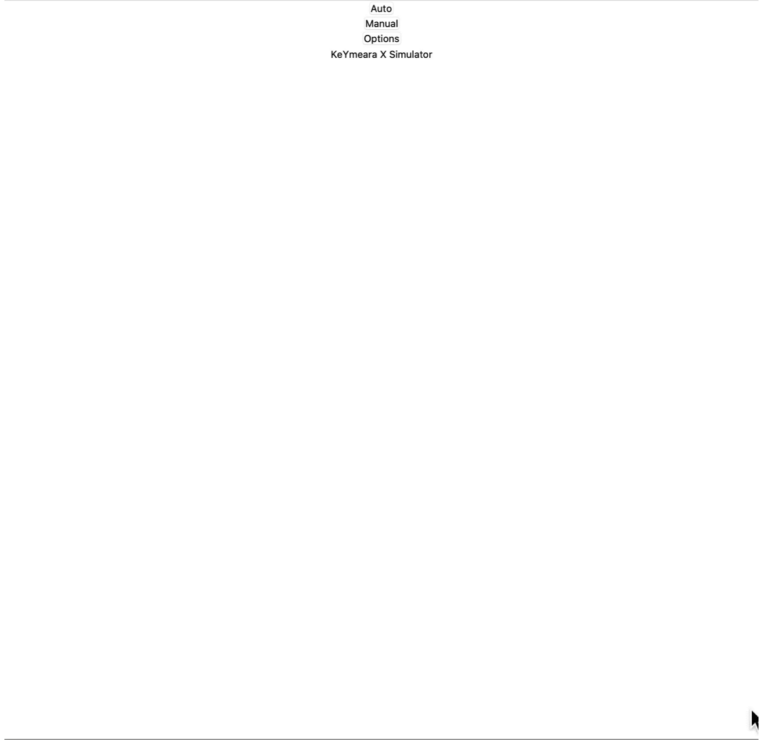
Auto mode: faulty lab 4 results in collision

Auto mode: safe lab 4, risky trajectory for correct controller

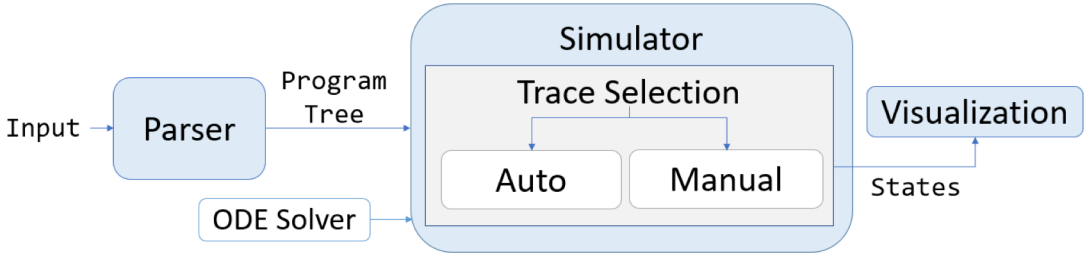


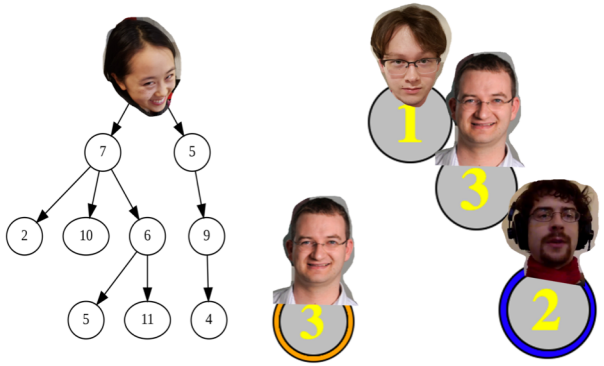
Auto Mode: broken lab 3

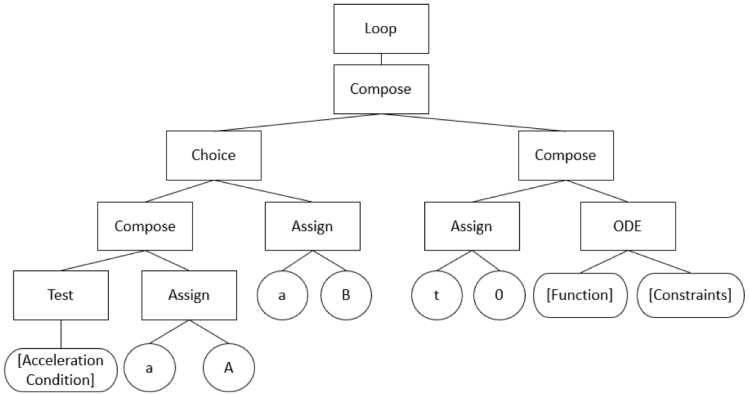
Auto mode: safe lab 3

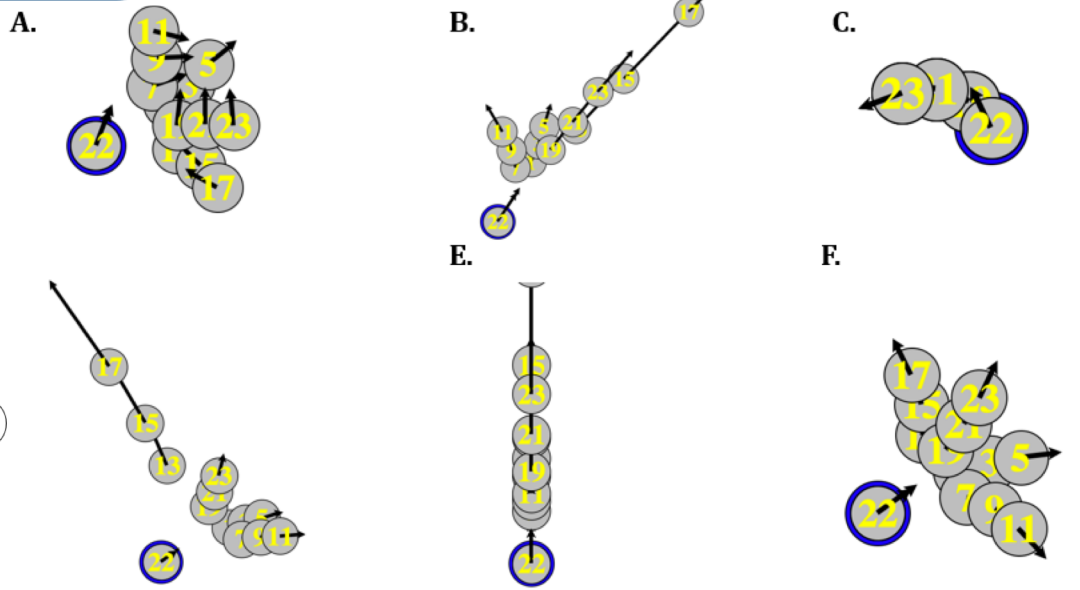


Questions?


Input
Parser
Program Tree
Simulator
Trace Selection
Auto
Manual
Visualization
States
ODE Solver




Loop
Compose
Choice
Compose
Compose
Assign
Assign
ODE
Test
Assign
a
B
t
0
[Function]
[Constraints]
[Acceleration Condition]
a
A

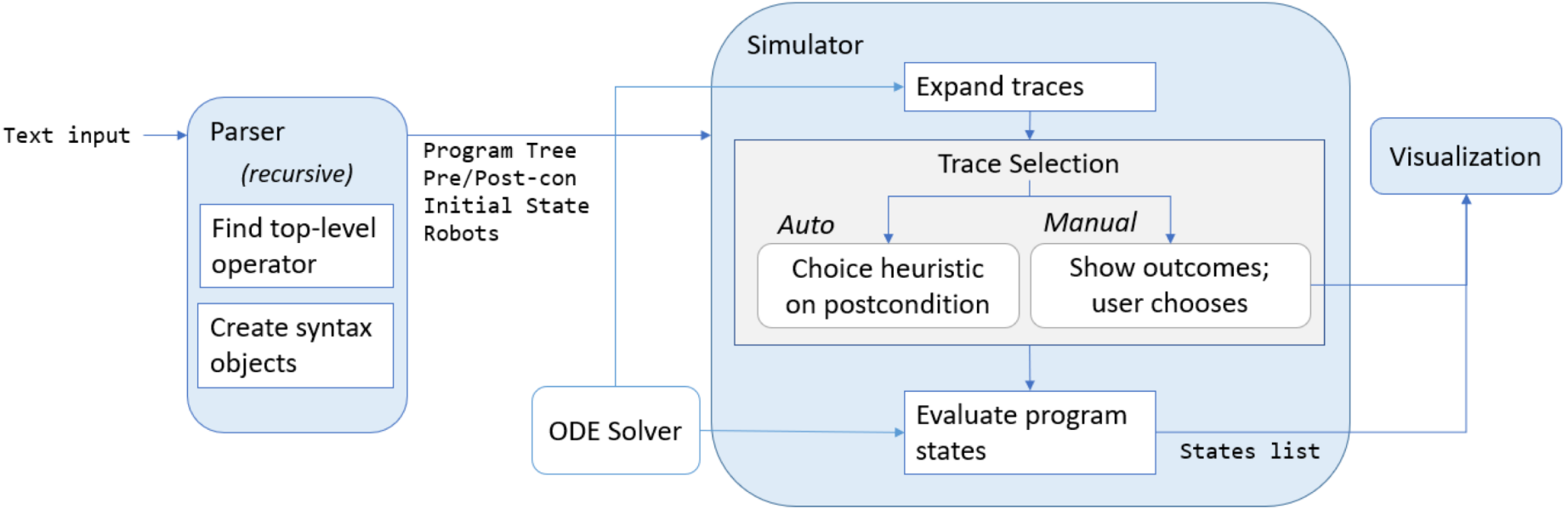

A.
B.
C.
E.
F.

Supplements

Future Work

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

System Architecture



Handling loops

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

$$((?\text{canAccelerate}; a:=A) \cup a:=B); t:=0; \{\text{ODE}\})^*$$

* Doesn’t allow for nested loops