Facilitating Concurrency in Hybrid Programs

DECEMBER 10, 2019

15-824: Logical Foundations of Cyber-Physical Systems (Fall 2019)

Haithem Turki
Long Pham

Languages shape our thinking

Shadenfreude

- Malicious enjoyment of the misfortunes of others. (Oxford English Dictionary)

Overview

- Motivation and Background
- Theory
- Implementation
- Future Work

Motivation and Background

Cyber-physical systems (CPSs) are inherently composite

Interaction between CPSs

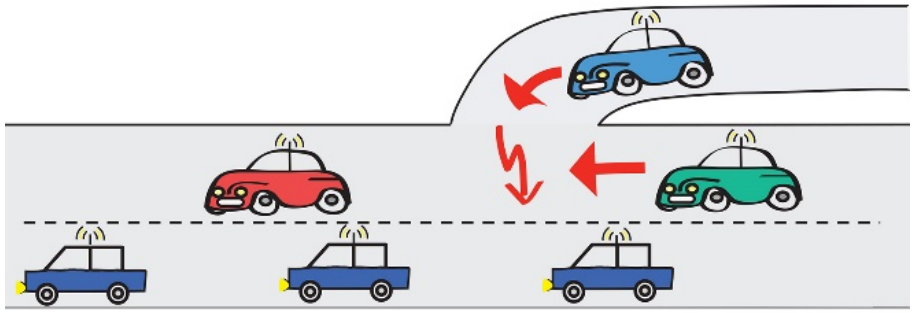


Image courtesy of http://lfcps.org/course/lfcps19.html

Interaction within a CPS

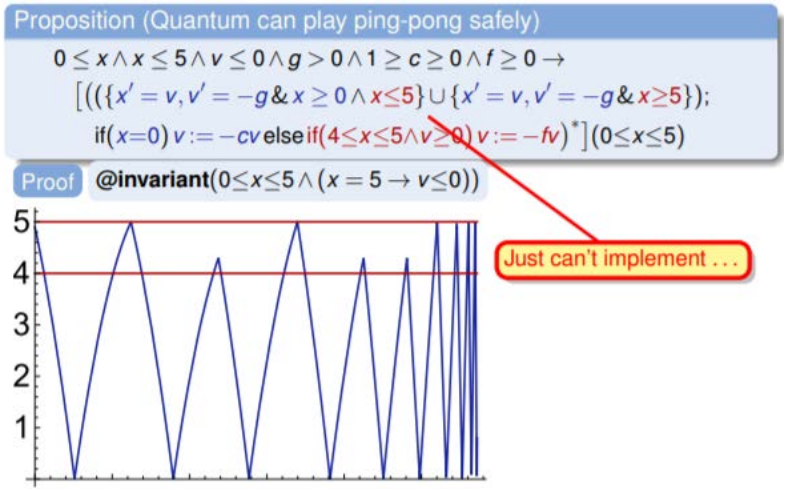


Image courtesy of https://lfcps.org/lfcps/slides/09-time-slides.pdf

Inputs, outputs, and parallel composition

C. A. R. (Tony) Hoare,
Communicating Sequential Processes, 1978

Image courtesy of Microsoft Research

Inputs, outputs, and parallel composition

Consumer = order!coffee -> how-much?x -> pay!x -> get?y -> STOP

Barista = order?y -> how-much!2 -> pay?x -> get!y -> STOP

Consumer || Barista = order!coffee -> how-much!2 -> pay!2 -> get!coffee -> STOP

Inputs, outputs, and parallel composition

Consumer = order!coffee -> how-much?x -> get?y -> pay!x -> STOP

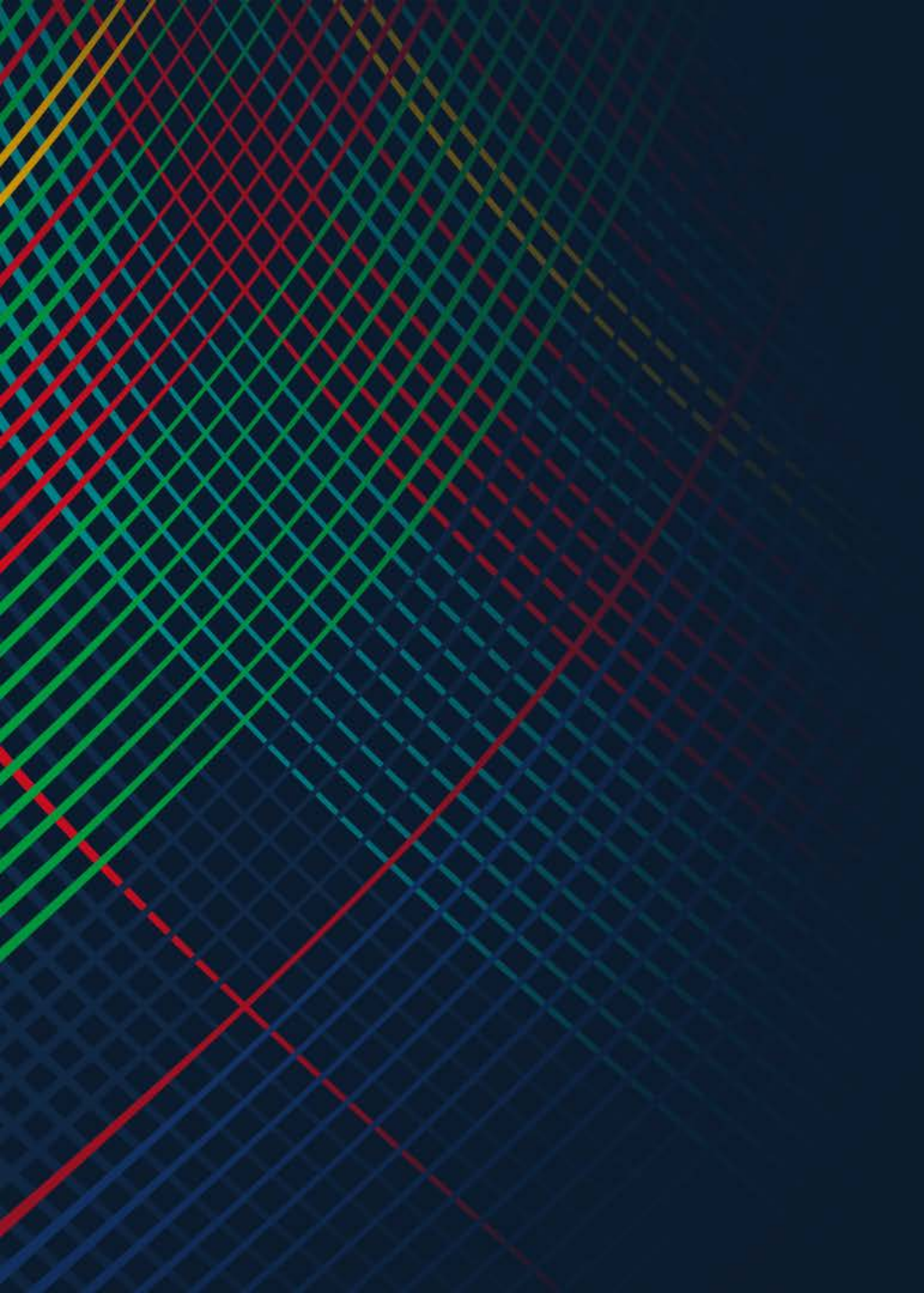
Barista = order?y -> how-much!2 -> pay?x -> get!y -> STOP

Consumer || Barista = order!coffee -> how-much!2 -> STOP

Adding concurrency to hybrid programs

Our Contributions

- Adding channels to hybrid programs
- Providing trace semantics
- Devising a sequentialization algorithm
- Extending KeYmaera X to provide a proof-of-concept implementation supporting our augmented syntax



Theory

Local variables

Consumer = x:=1; pay!x; x:=0

Barista = pay?x; x:=x+1

Global variables

$$\begin{aligned}\{t' = -1\} \,||\, \{t' = -3\} = {} & \{t' = -1\}; \{t' = -4\}; \{t' = -3\} \\ & \cup \{t' = -1\}; \{t' = -4\}; \{t' = -1\} \\ & \cup \{t' = -3\}; \{t' = -4\}; \{t' = -1\} \\ & \cup \{t' = -3\}; \{t' = -4\}; \{t' = -3\}.\end{aligned}$$

Syntax of concurrent hybrid programs

$\alpha, \beta ::= x := e$	assignment
$\mid ?Q$	test
$\mid x' = f(x) \,\&\, Q$	ordinary differential equation; ODE
$\mid \alpha \cup \beta$	nondeterministic choice
$\mid \alpha; \beta$	sequential composition
$\mid \alpha^*$	nondeterministic finite repetition
$\mid \alpha \,\|\|\, \beta$	parallel composition
$\mid c!e$	blocking channel write
$\mid c?x$	blocking channel read

Trace semantics

Definition 3.2 (Action). *An action is one of the following:*

1. *Assignment:* $x := e$;
2. *Test:* $?Q$;
3. *Ordinary differential equation:* $(\{x' = f(x) \,\&\, Q\}, r)$, *where* $r \in \mathbb{R}$ *denotes the duration of continuous evolution;*
4. *Input:* $c?x$, *where* $c \in \mathcal{C}$ *is a channel name and* $x \in \mathcal{V} = \mathcal{V}_\ell \cup \mathcal{V}_g$ *is a variable symbol.*
5. *Output:* $c!e$, *where* $c \in \mathcal{C}$ *is a channel name and* e *is a polynomial term.*

Trace semantics

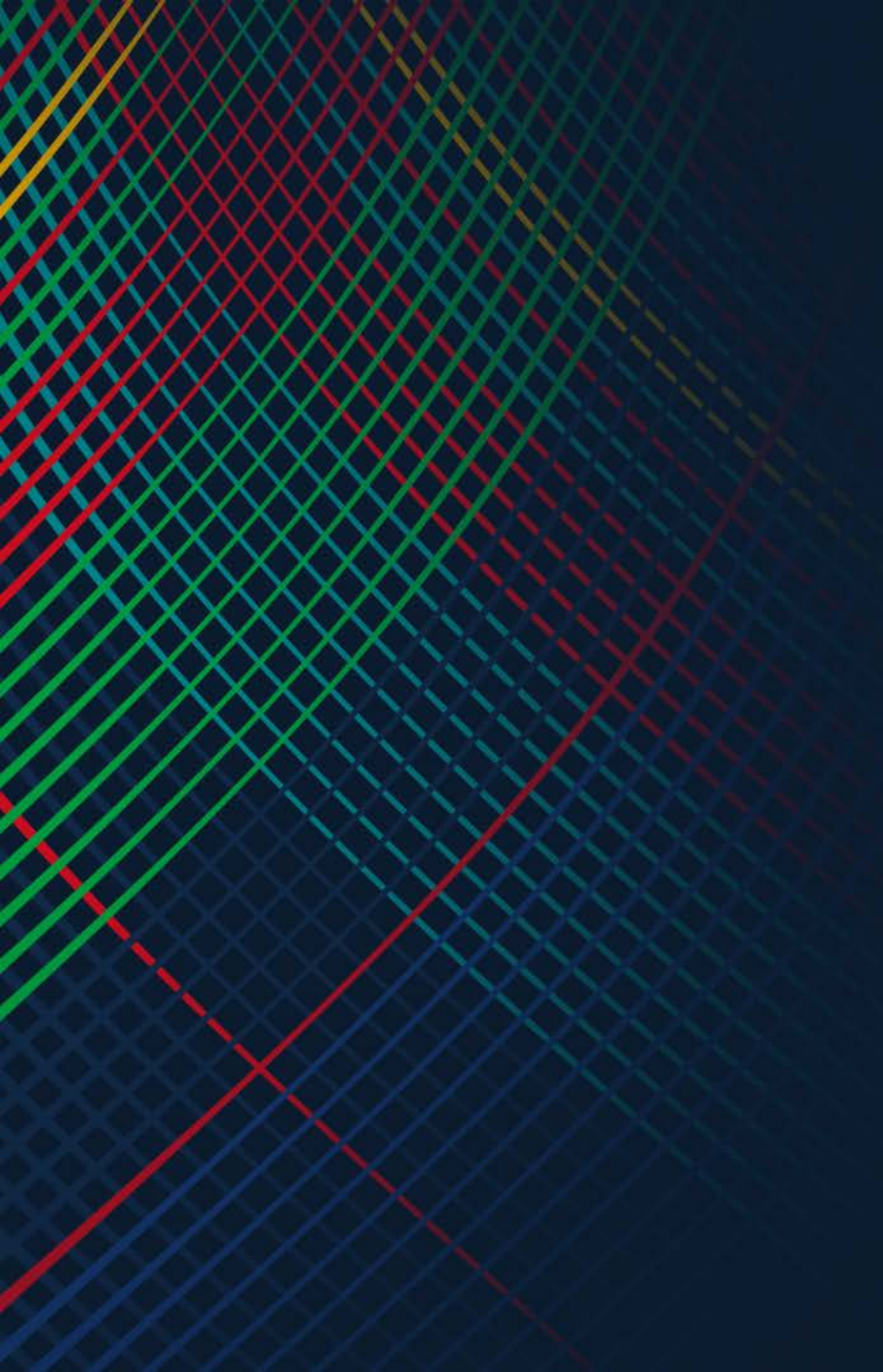
Definition 3.5 (Interleaving of sequences of actions)**.** *Suppose we are given a state* $v \in State(\mathcal{V})$ *and two sequences* $\sigma = (\sigma_1, \ldots, \sigma_n)$ *and* $\rho = (\rho_1, \ldots, \rho_m)$ *of actions. The interleaving of* σ *and* ρ *with respect to* v*, denoted by* $v \vdash (\sigma \,||\, \rho)$*, is a set of sequences of actions inductively defined below. For brevity, whenever there is symmetry, we only present one of two dual cases.*

Trace semantics

4. *If* $\sigma_1 = c_1!e$ *and* $\rho_1 = c_2?x$, *then*

$$v \vdash \sigma \,||\, \rho = c_1!u \circ (v' \vdash (\sigma_2, \ldots, \sigma_n) \,||\, (\rho_2, \ldots, \rho_n)),$$

where $c_1 \equiv c_2$, $u = v[\![e]\!]$, *and* v' *is the result of applying* $[x := u]$ *to state* v.



Implementation

Current Implementation

Authoring Parallel Programs

- Extended existing KeYmaera X [1] theorem prover
- Augmented KeYmaera X parser and lexer to handle necessary syntax
- Added new *Parallel* program type
- Propagated new *Channel* concept throughout existing KeYmaera X codebase

airConditioner

```
Definitions
  Real Thot;
  Real Tcold;
End.

ProgramVariables
  Real t;
End.

Problem
  (Thot > Tcold)
  ->
  <{{{?(t > Thot); {t'=-1}} ++ {?(t < Tcold); {t'=3}}} || {{?(t > Thot); {t'=-3}} ++ {?(t < Tcold); {t'=1}}} & t}>
  (t <= Thot & t >= Tcold)
End.
```

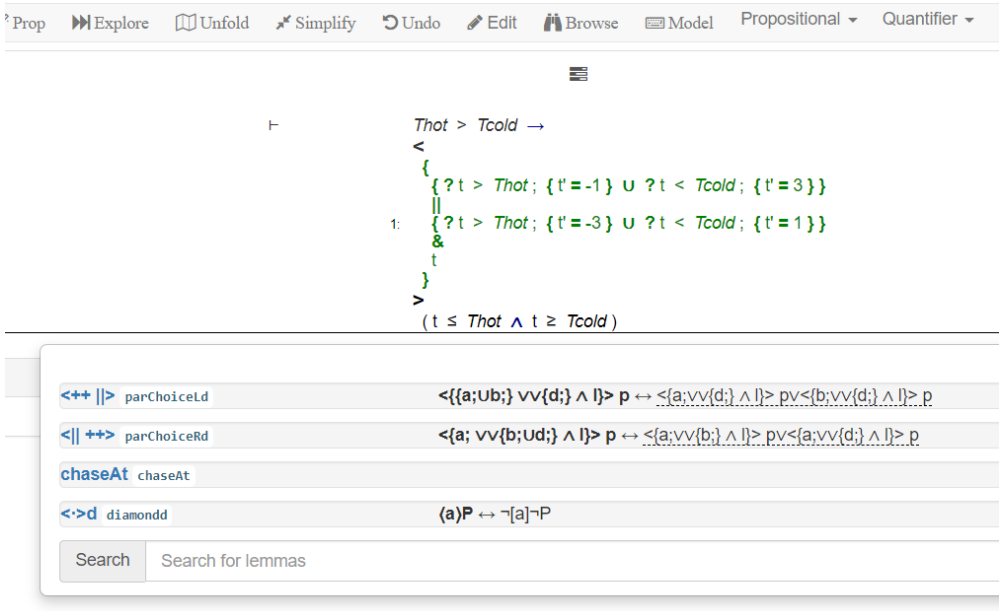
Authoring parallel programs in KeYmaera X web interface

[1] http://www.ls.cs.cmu.edu/KeYmaeraX/

Current Implementation

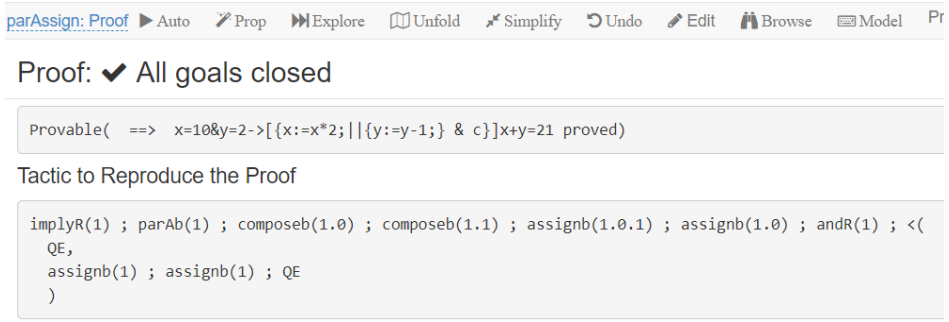
Proofs for Parallel Programs

- Implemented subset of the core and derived axioms needed to prove the safety of parallel programs
- Full set of axioms still to be implemented
- But able to complete proofs of small examples in current implementation

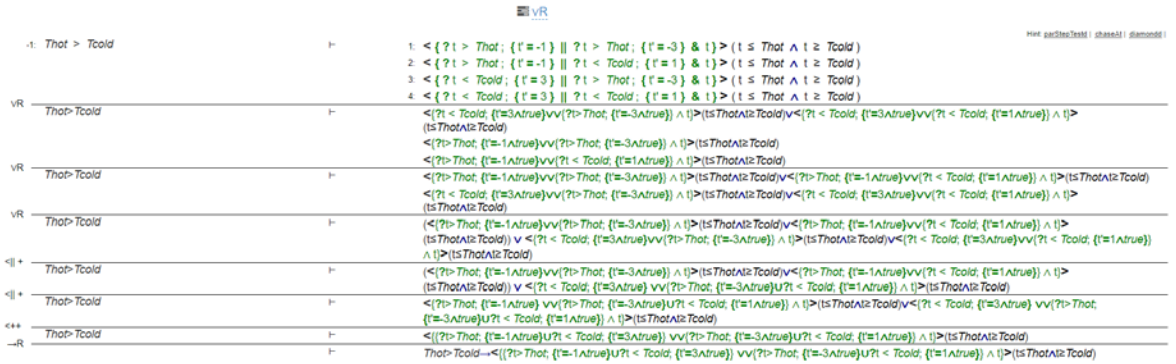


Parallel axioms suggested in KeYmaera X web interface

Current Implementation



Completed proof in KeYmaera X web interface



Proof tree in KeYmaera X web interface

Future Work

- Providing full implementation of our extension in KeYmaera X
- Providing a formal definition of equality based on trace semantics
- Formally proving the sequentialization axioms
- Introduce more constructs to make calculus for concurrent hybrid programs more expressive
  - Example: wake up a stalled program when an ODE of another program reaches a certain state

Thank you!

- Code available on GitHub: https://github.com/hturki/KeYmaeraX-release/tree/parallel