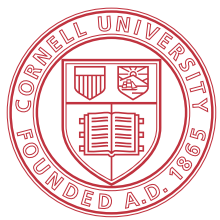
ROSCoq: Robots powered by Constructive Reals

Abhishek Anand
(joint work with Ross Knepper)

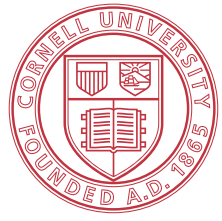


Cornell University

ITP, 27-Aug-2015, Nanjing

ROSCoq: Robots powered by Constructive Reals

Abhishek Anand
(joint work with Ross Knepper ⇑ [1])



Cornell University

ITP, 27-Aug-2015, Nanjing

[1] The symbol ⇑ occurs at many places in this PDF. On a modern PDF reader (with support for OCG) such as Adobe Reader, Evince (Linux), Foxit Reader (Windows), PDF XChange viewer clicking such a symbol will open/close a popup containing extra explanation, e.g. what I said while describing that slide)

Motivation

Collaborating Robots[2] ⇑
Keyboard finding robot[3] ⇑

[2] Mehmet Dogar, Ross A. Knepper, Andrew Spielberg, Changhyun Choi, Henrik I. Christensen, and Daniela Rus. "Towards Coordinated Precision Assembly with Robot Teams". In: *ISER*. 2014.

[3] Abhishek Anand, Hema Swetha Koppula, Thorsten Joachims, and Ashutosh Saxena. "Contextually guided semantic labeling and search for three-dimensional point clouds". In: *IJRR* (2012).

Goals

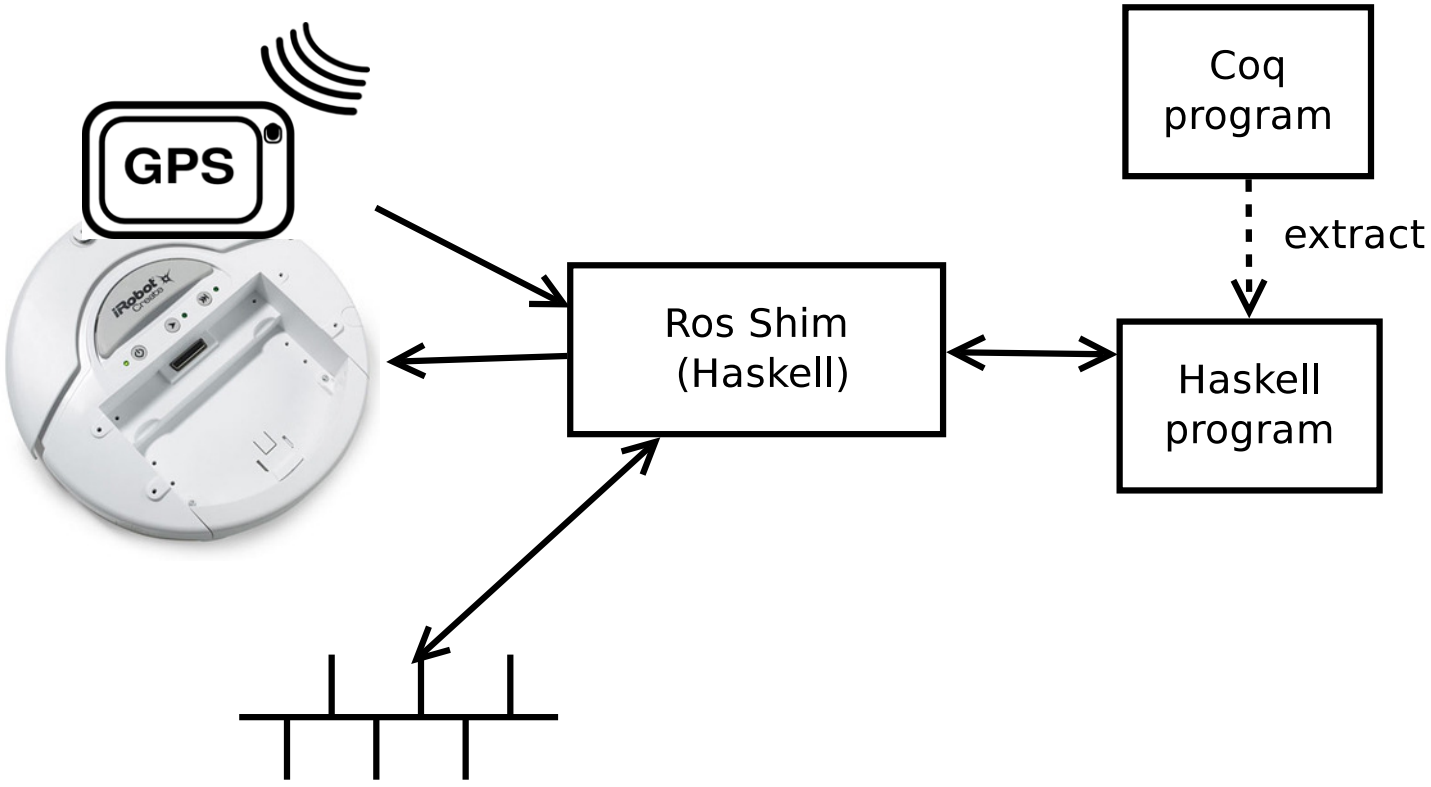
- Write robotic programs in Coq

Goals

- Write robotic programs in Coq
- “Run” them on actual robots using a shim:

Goals

- Write robotic programs in Coq
- "Run" them on actual robots using a shim:



- Specify the behavior of the shim, physics, and hardware in a *realistic* way

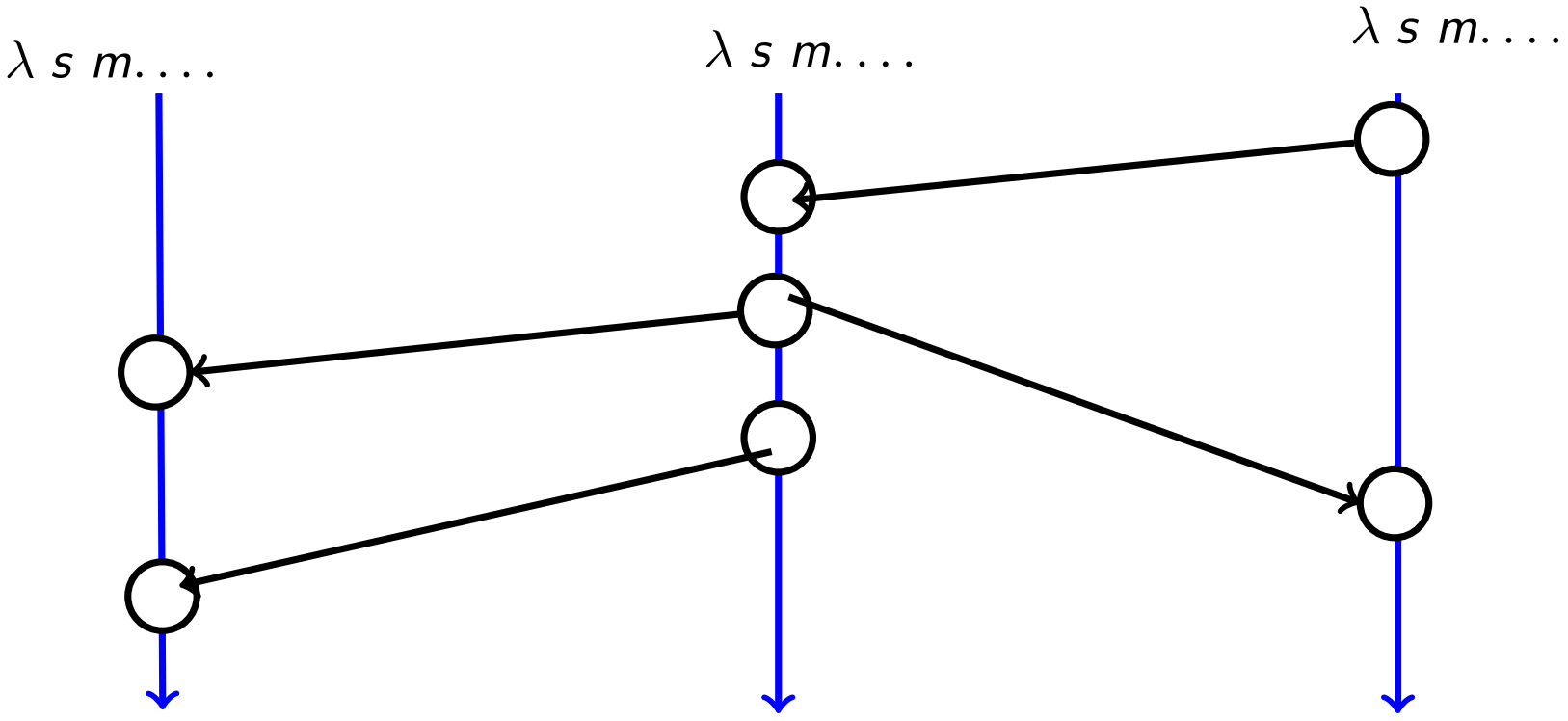
Goals

- Write robotic programs in Coq
- "Run" them on actual robots using a shim:
- Specify the behavior of the shim, physics, and hardware in a *realistic* way
- develop Coq proofs of properties about the overall behavior

The Logic of Events framework

- Cyber Physical systems are distributed systems
- where agents have fancy sensing and/or actuation actuation capabilities.

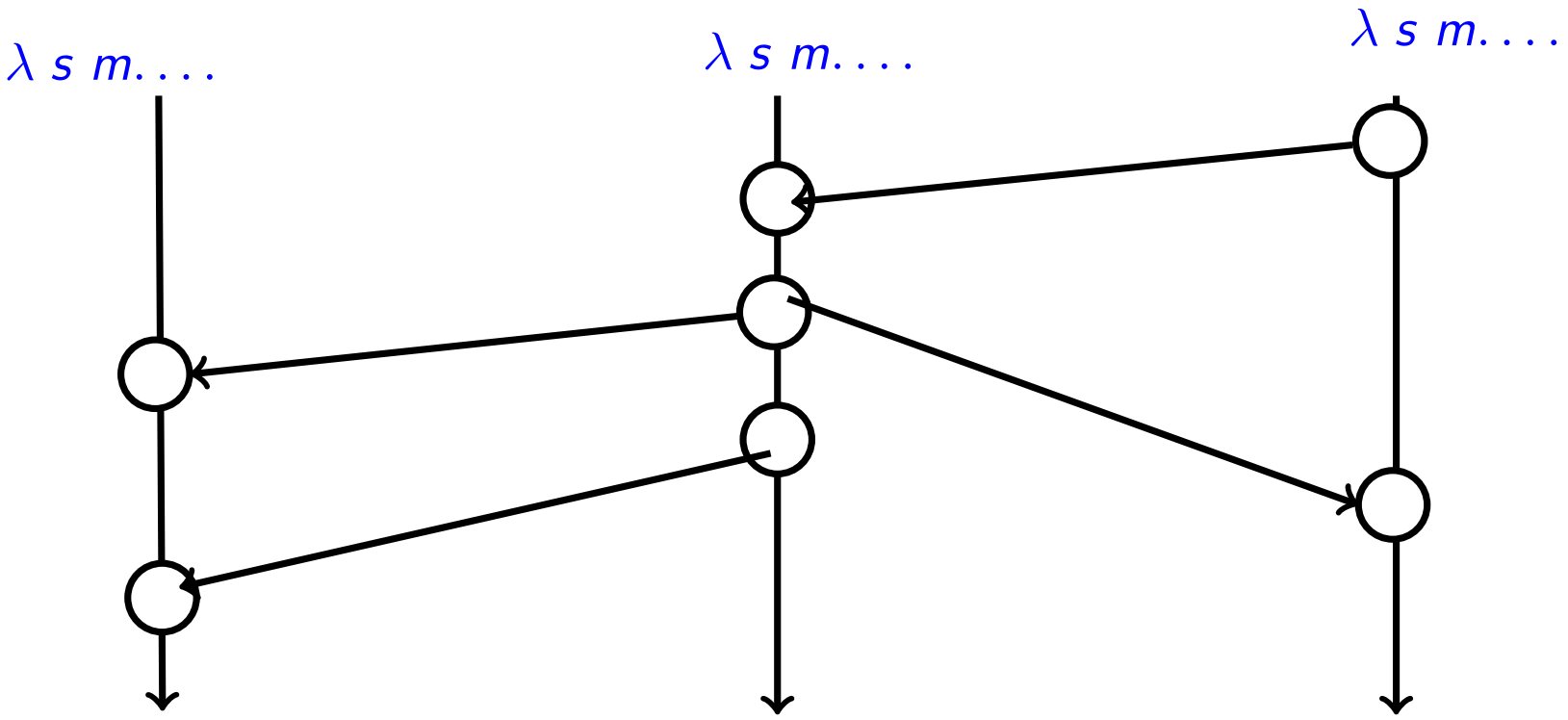
The Logic of Events framework



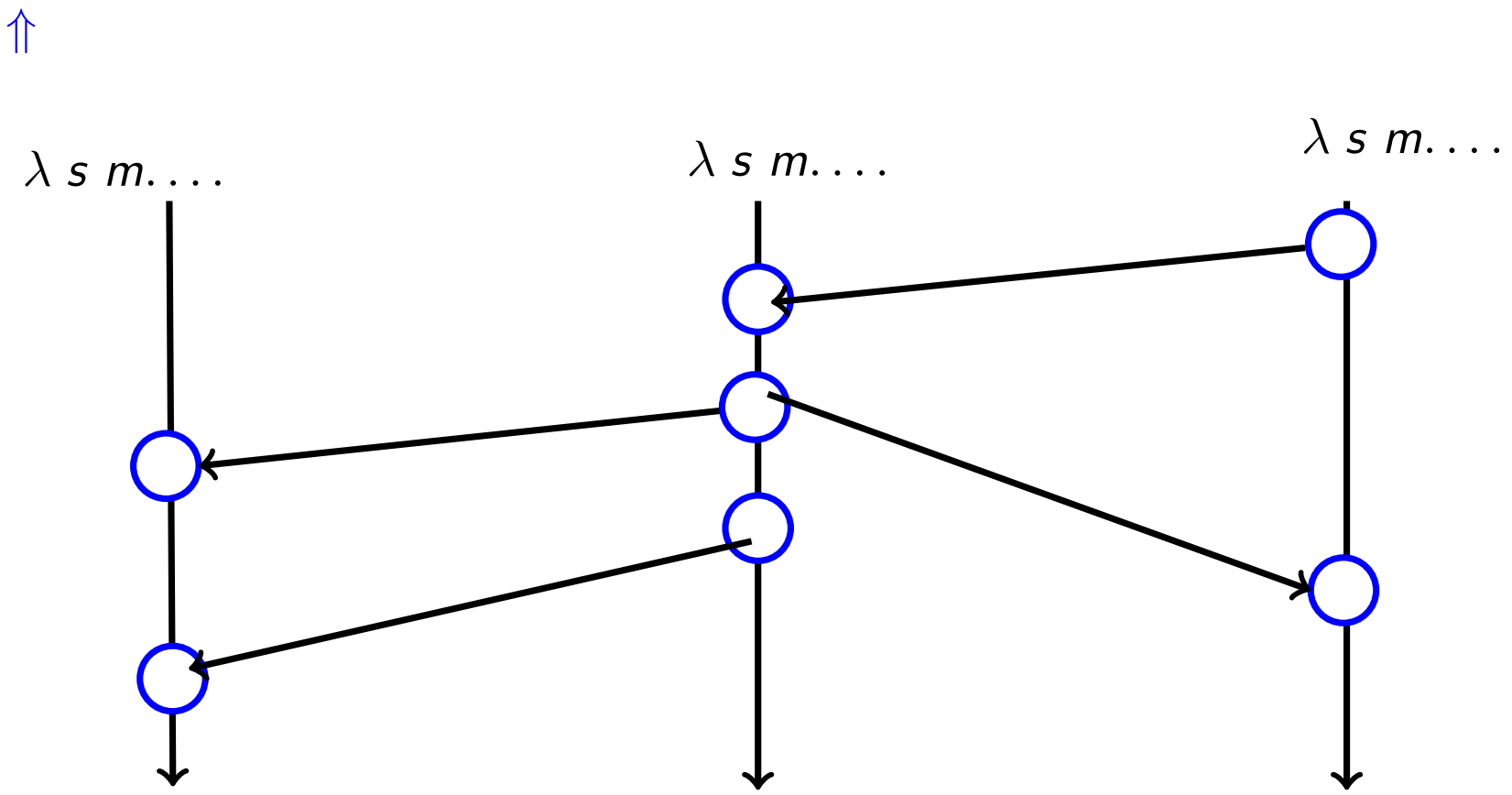
Nicolas Schiper, Vincent Rahli, Robbert Van Renesse, Marck Bickford, and Robert L. Constable. "Developing correctly replicated databases using formal tools". In: *DSN*. IEEE, 2014, pp. 395–406 ⇑

The Logic of Events framework

$\Uparrow$

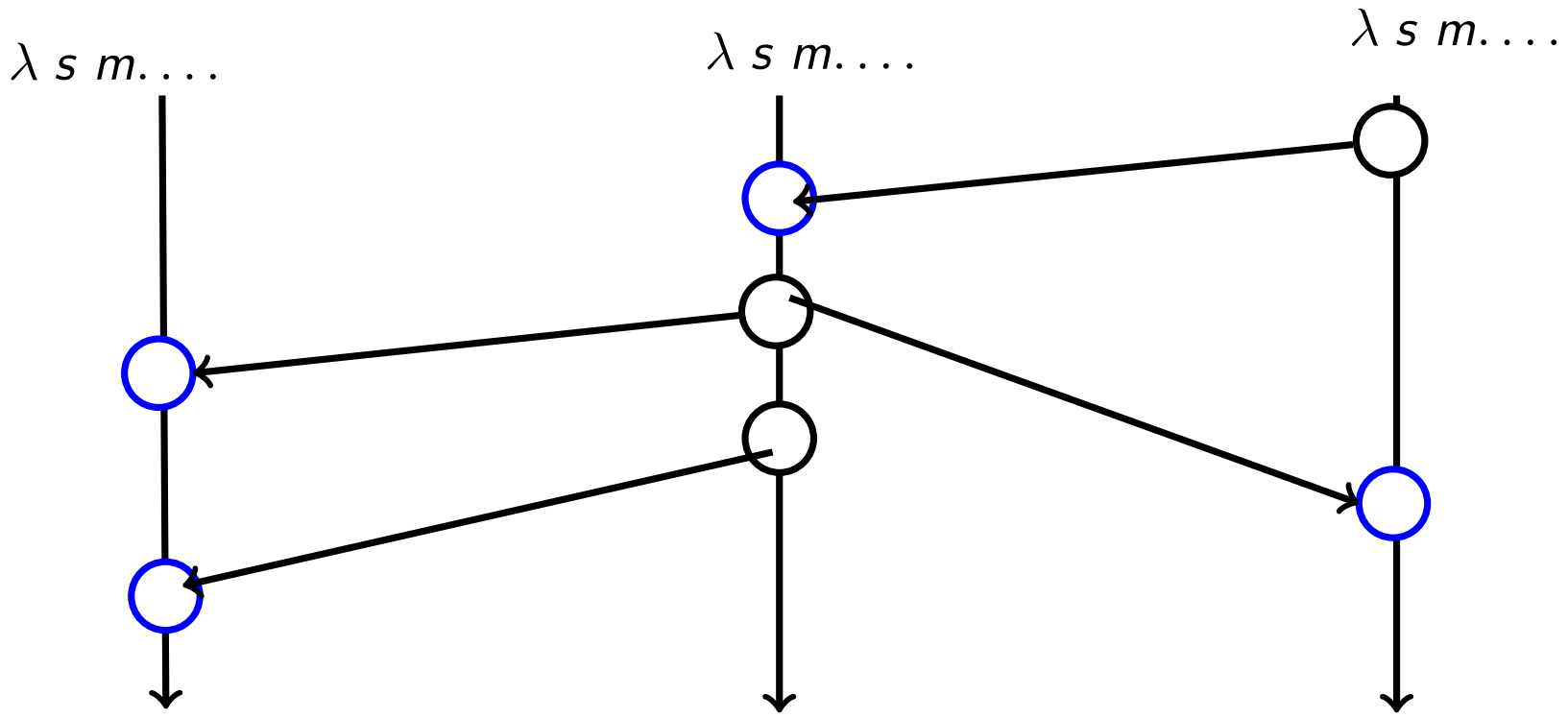


The Logic of Events framework

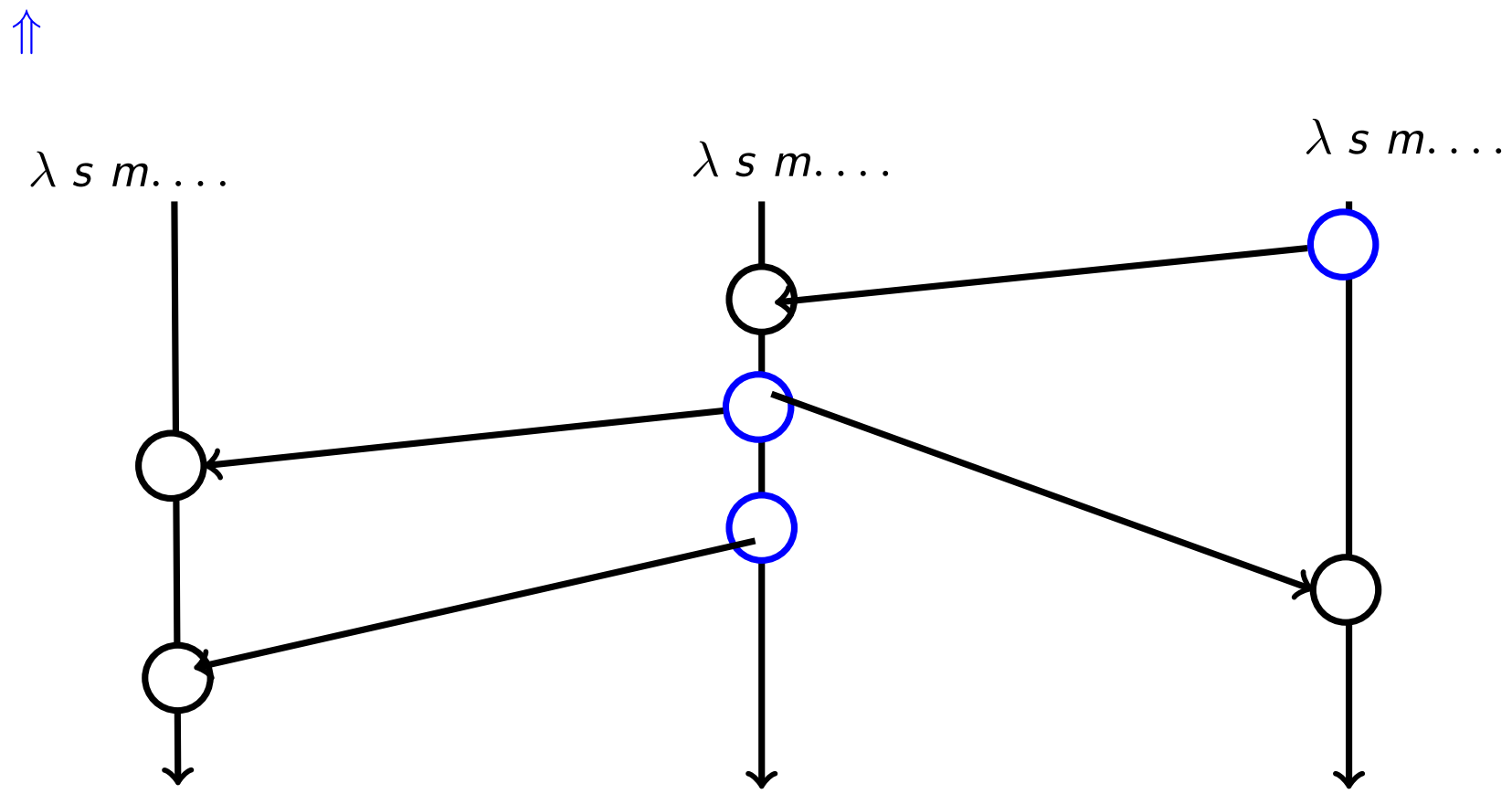


The Logic of Events framework

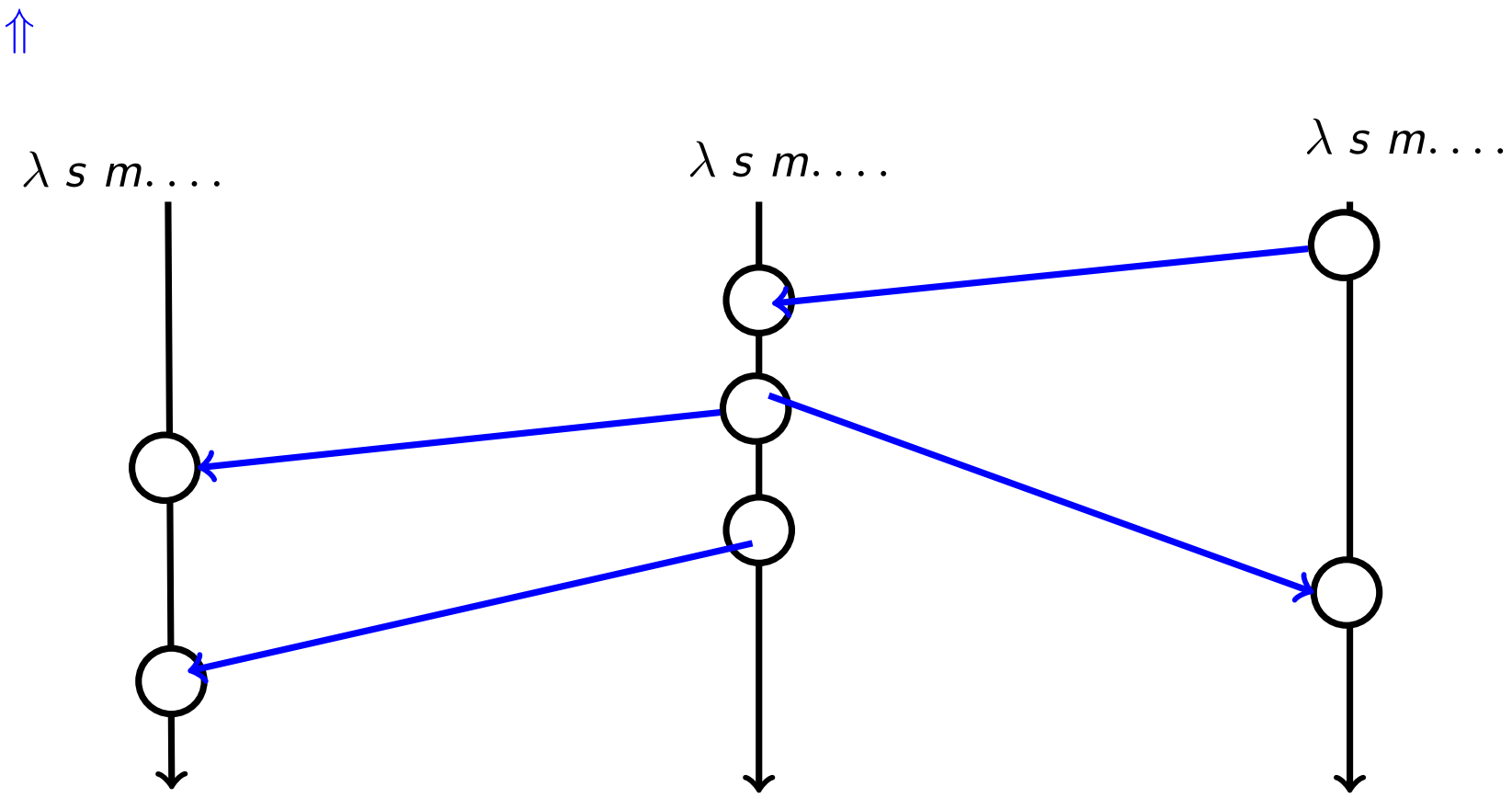
⇑



The Logic of Events framework

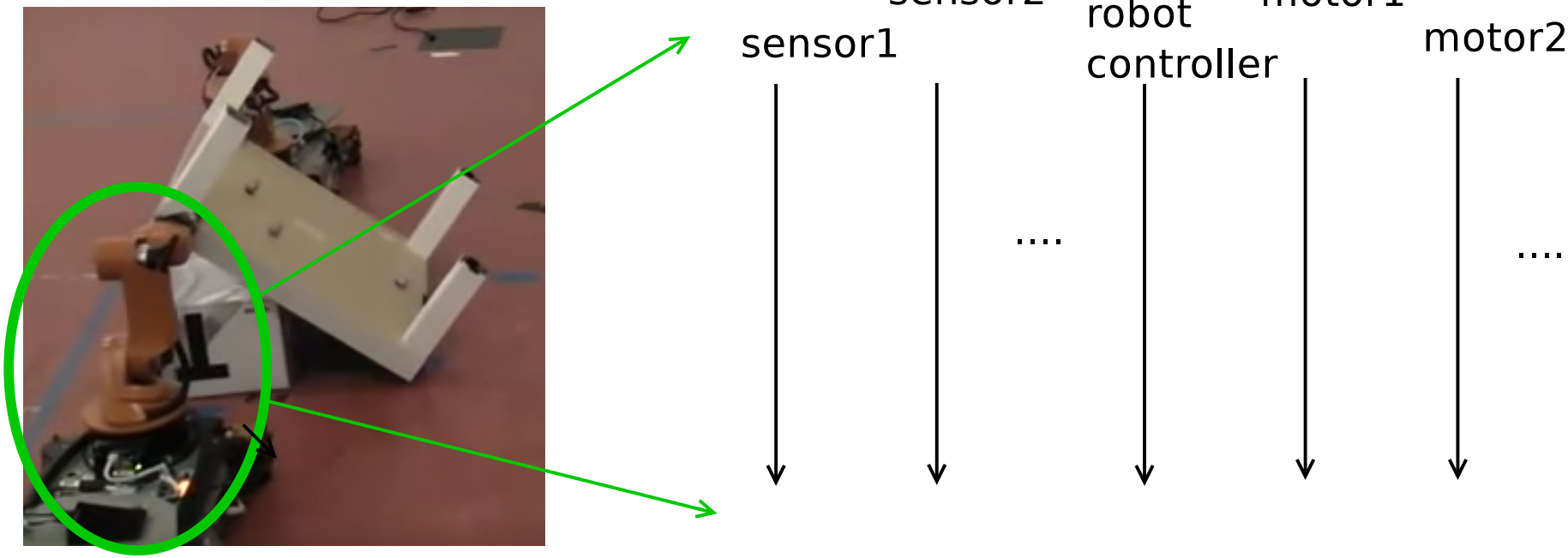


The Logic of Events framework



ROS : Even single robots are distributed systems

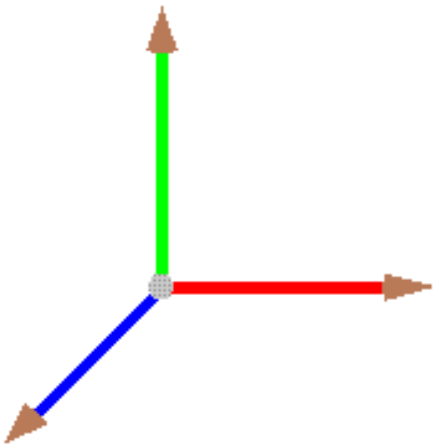
- Based on asynchronous message passing[4]
- Very popular, drivers (as DS agents) available for many robots [5]

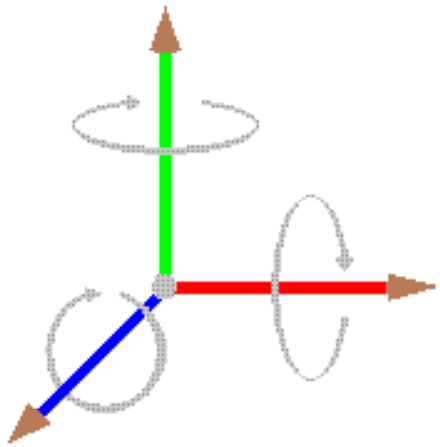


[4] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software*. Vol. 3. 2009, p. 5.

ROS : Even single robots are distributed systems

- Based on asynchronous message passing[4]
- Very popular, drivers (as DS agents) available for many robots [5]
- uniform API ⇑





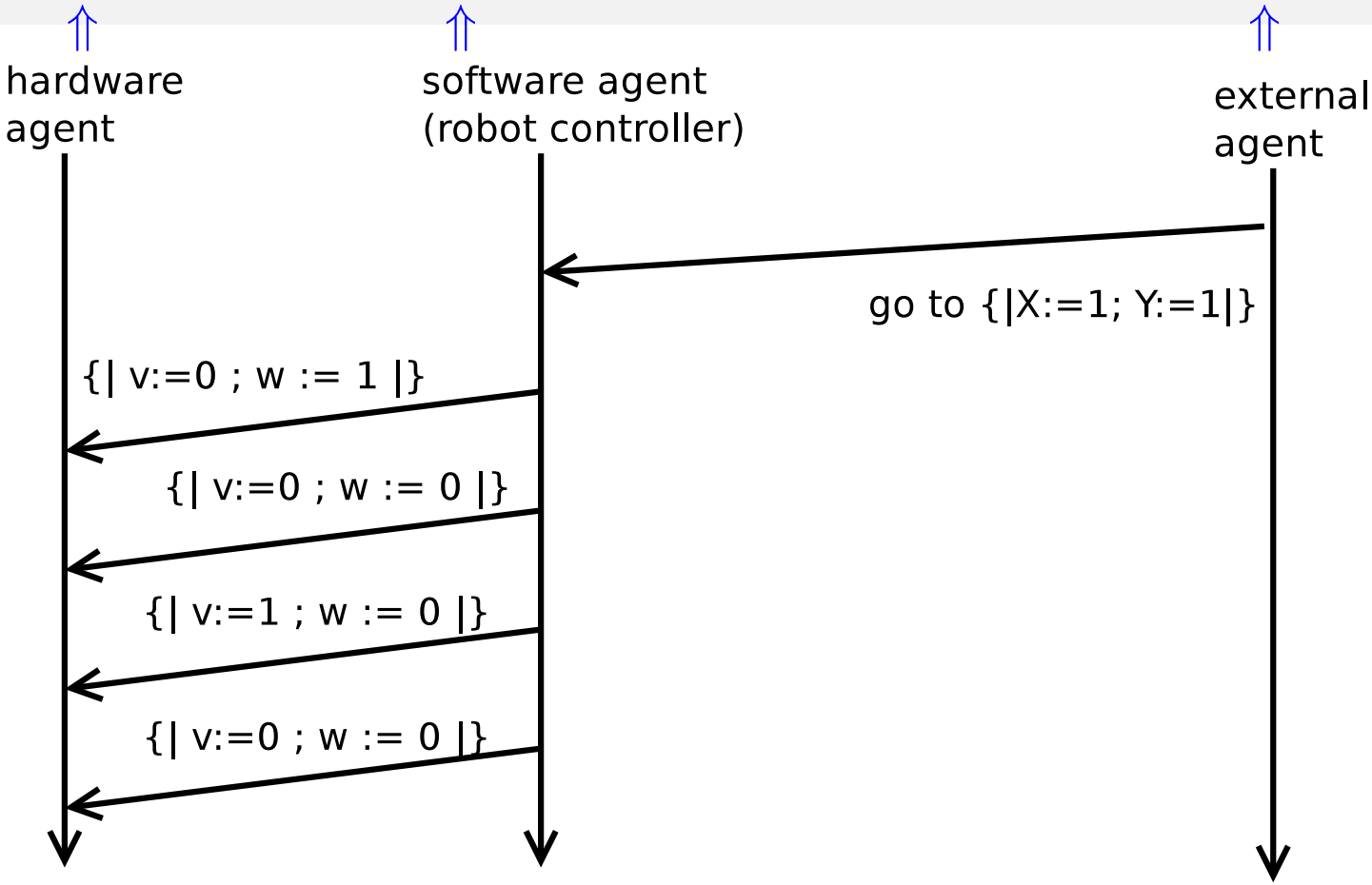
ROS : Even single robots are distributed systems

- Based on asynchronous message passing[4]
- Very popular, drivers (as DS agents) available for many robots [5]
- uniform API
- Running Example

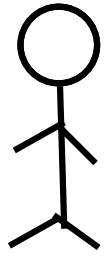
[4] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software*. Vol. 3. 2009, p. 5.

[5] http://wiki.ros.org/Robots

Running Example



```
Definition robotPureProgam (target : Cart2D Q) : list (Q × Polar2D Q) :=
  let polarTarget : Polar2D ℝ := Cart2Polar target in
  let rotDuration : ℝ := | θ polarTarget | / rotspeed in
  let translDuration : ℝ := (rad polarTarget) / speed in
  [ (0,{| rad:= 0 ; θ := ( polarθSign target ) * rotspeed |})
      ; ( tapprox rotDuration delRes delEps , {| rad := 0 ; θ := 0 |})
      ; (delay , {| rad :=  speed ; θ := 0 |})
      ; ( tapprox translDuration delRes delEps , {| rad := 0 ; θ := 0 |}) ].
```

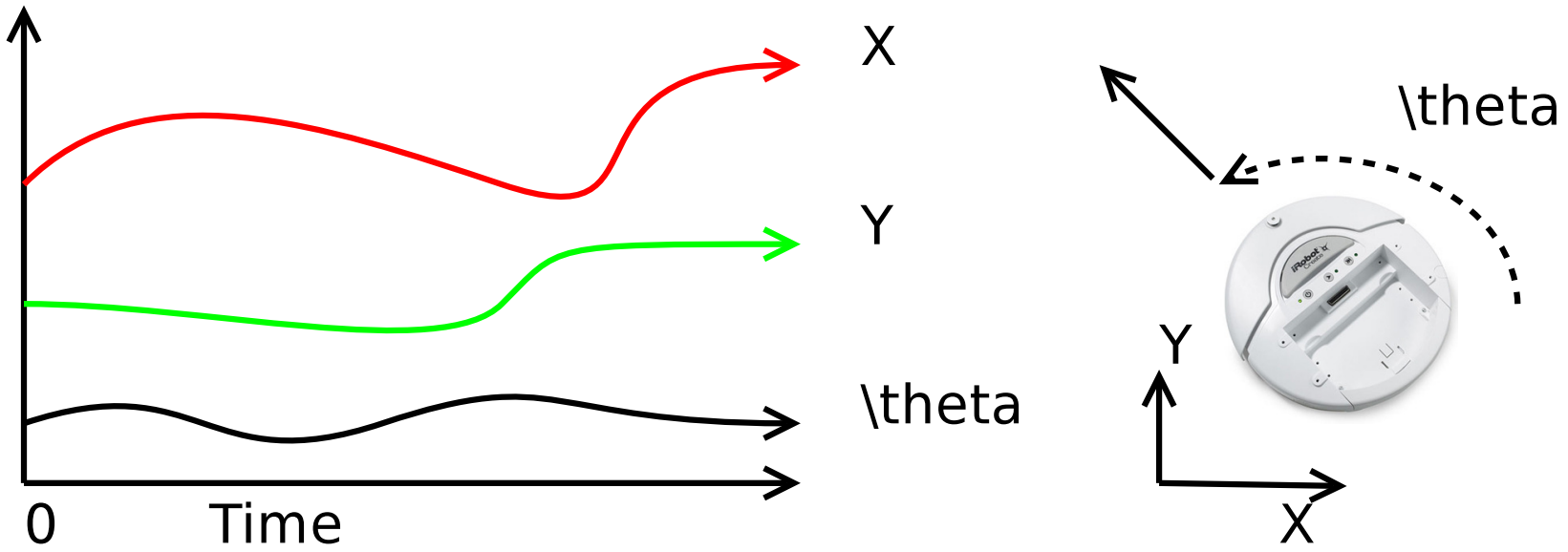


How to Specify a CPS in ROSCoq

- Define the physical model as a Coq type
- Define the collection of agents
- Specify the behavior of each agent
  - S/w agent : Coq program
  - H/w agent : ...

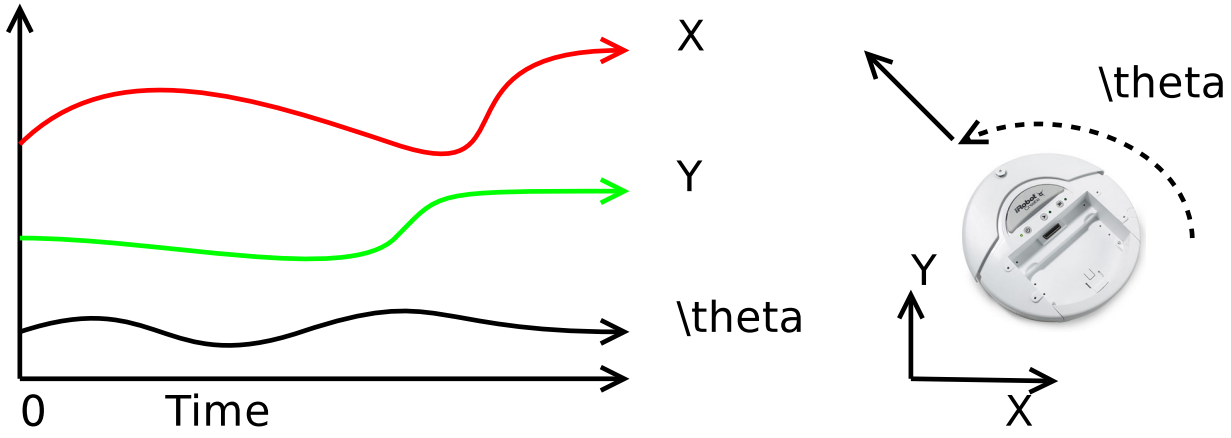
Physical Model of a CPS

Describes how the relevant physical quantities evolve over time



Physical Model of a CPS

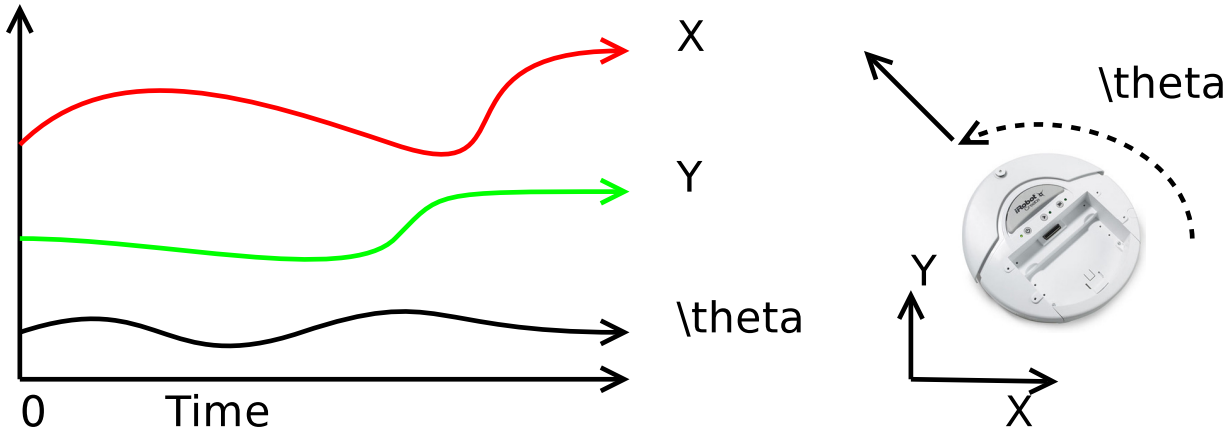
Describes how the relevant physical quantities evolve over time




```
Record Cart2D (T : Type) : Type := {X : T; Y: T}.

Record iCreate : Type := {
  position : Cart2D (Time →C ℝ);
  theta : (Time →C ℝ);
```

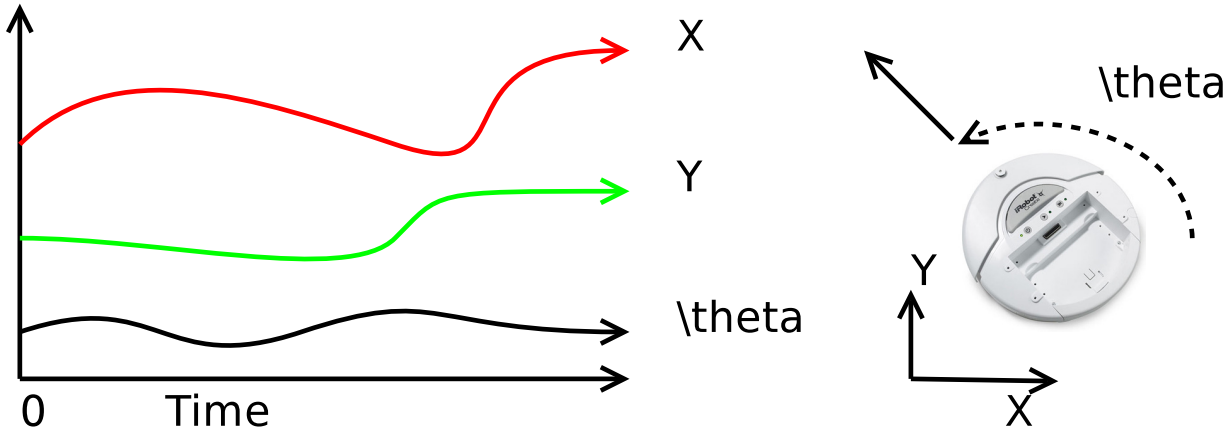
Physical Model of a CPS

Describes how the relevant physical quantities evolve over time




```
Record iCreate : Type := {
  position : Cart2D (Time →C ℝ);
  theta : (Time →C ℝ);
  linVel : (Time →C ℝ);
  omega : (Time →C ℝ);

  derivRot : isDerivativeOf omega theta;
  derivX : isDerivativeOf (linVel * ( FCos theta)) (X position);
  derivY : isDerivativeOf (linVel * ( FSin theta)) (Y position);
```

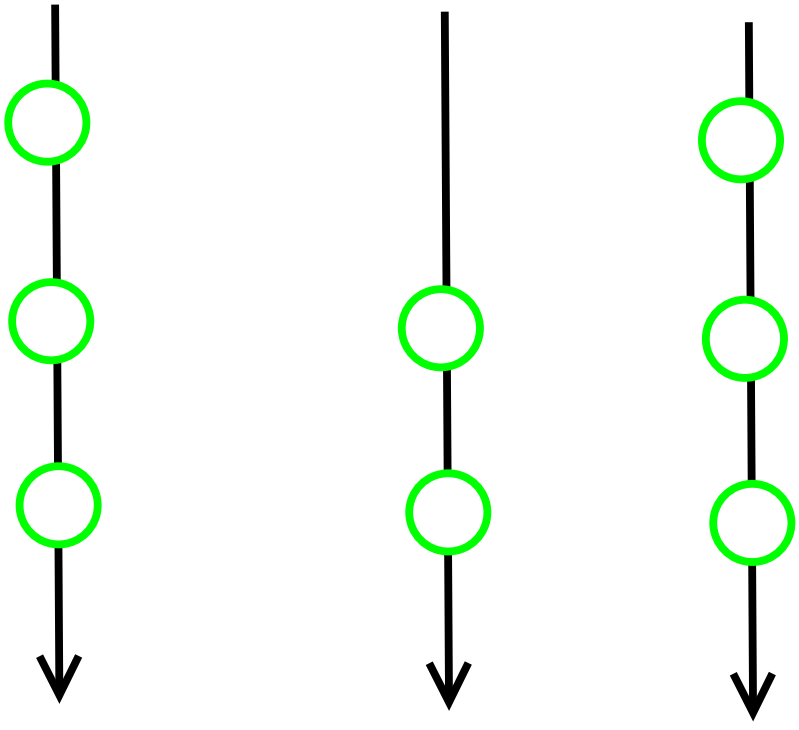
Physical Model of a CPS



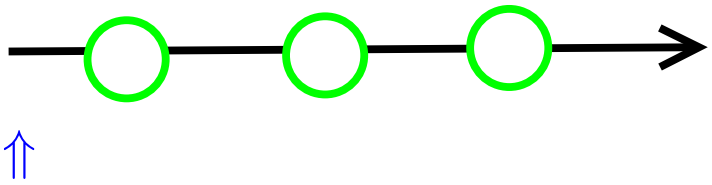

```
Record iCreate : Type := {
  position : Cart2D (Time →C ℝ);
  theta : (Time →C ℝ);
  linVel : (Time →C ℝ);
  omega : (Time →C ℝ);

  derivRot : isDerivativeOf omega theta;
  derivX : isDerivativeOf (linVel * ( FCos theta)) (X position);
  derivY : isDerivativeOf (linVel * ( FSin theta)) (Y position);
```

In case there were 2 robots, we would use iCreate × iCreate

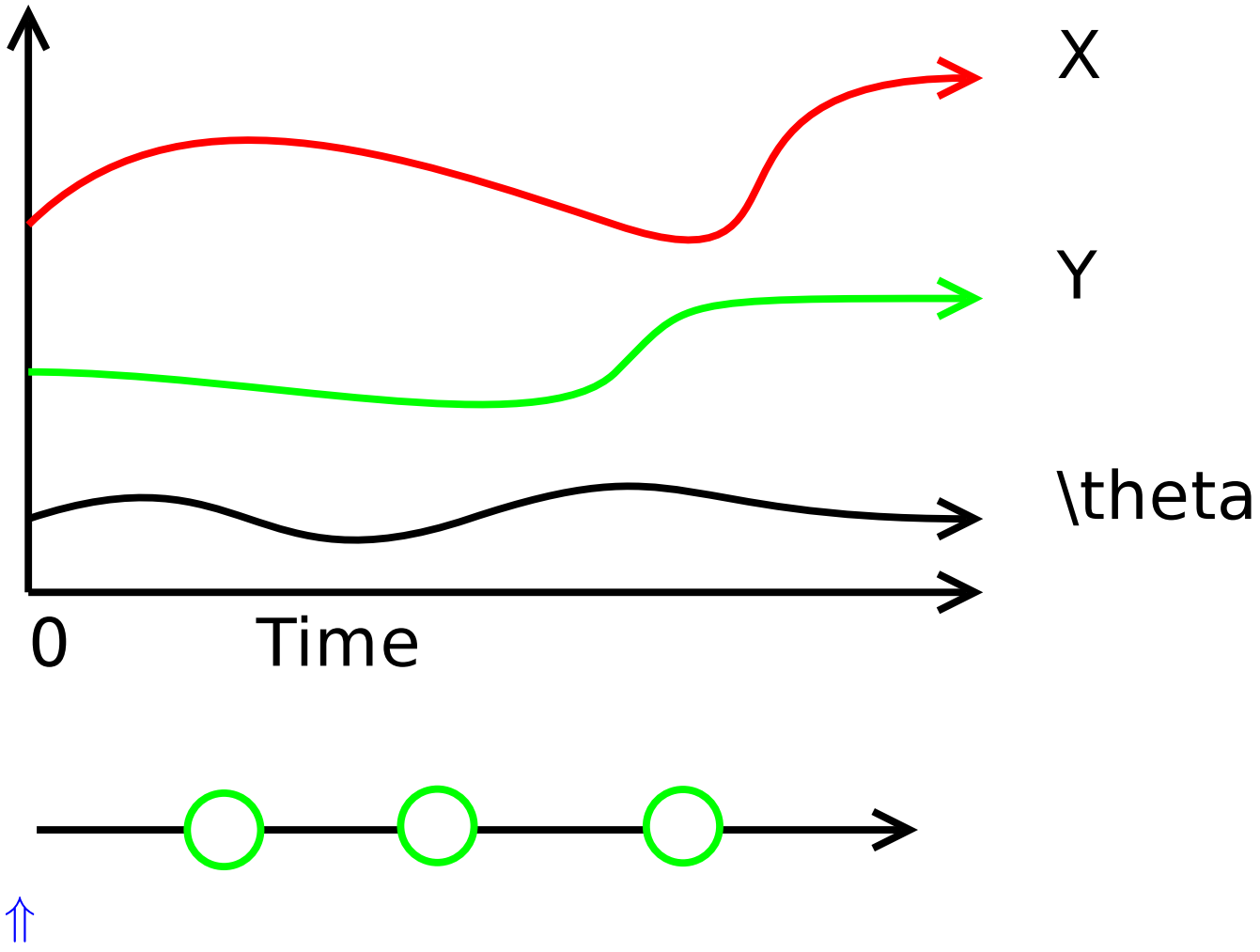
Semantics of Agents



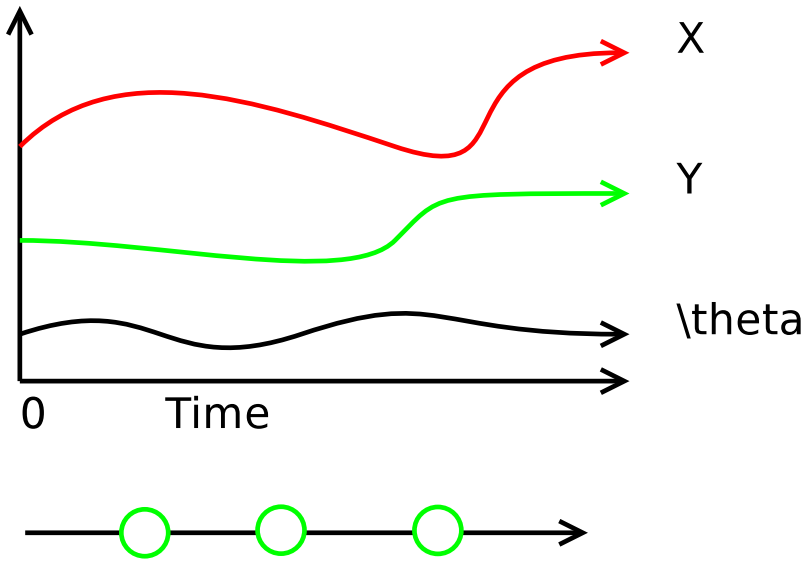
Semantics of Agents



Semantics of Agents

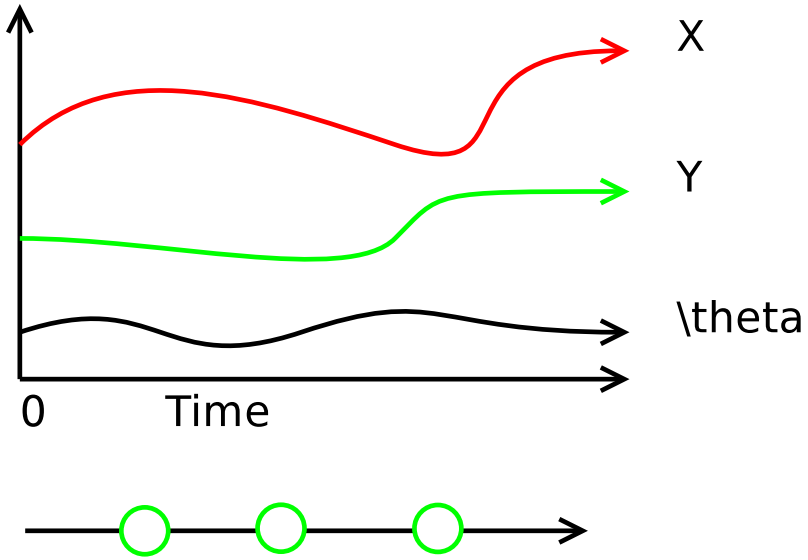


Semantics of Agents



- $PhysModelType \rightarrow (\mathbb{N} \rightarrow \text{option } Event) \rightarrow \texttt{Prop}$

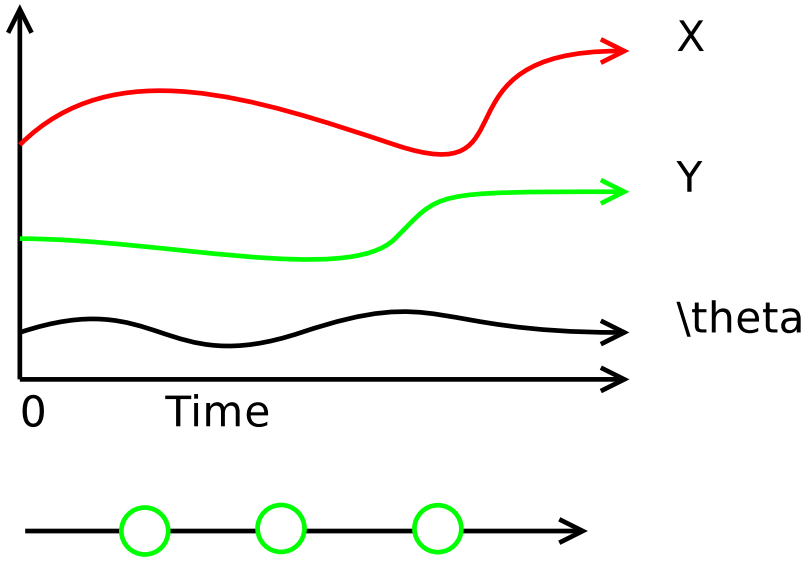
Semantics of Agents



- *PhysModelType* → (ℕ → option *Event*) → `Prop`
- iCreate → (ℕ → option *Event*) → `Prop` ⇑

```
Record iCreate : Type := {
  position : Cart2D (Time →C ℝ);
  theta : (Time →C ℝ);
  linVel : (Time →C ℝ);
  omega : (Time →C ℝ);
  ⋮
```

Semantics of Agents



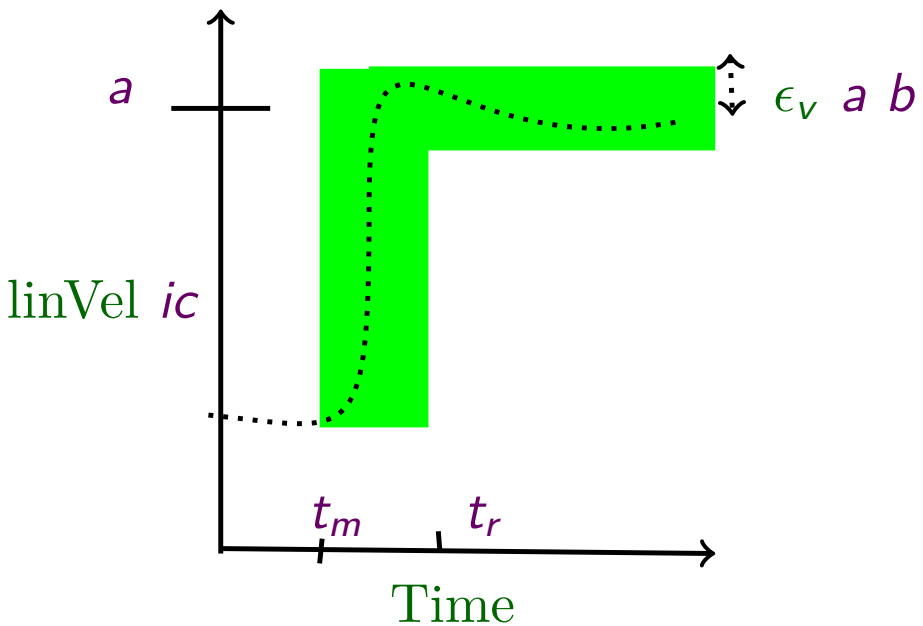
- Can handle non-determinstic devices ⇑
- Uniform treatment of both sensing and actuation devices ⇑

Specification of an iCreate (hardware agent)

```
Definition HwAgent  (ic: iCreate) (evs : nat → option Event): Prop :=
```

Specification of an iCreate (hardware agent)

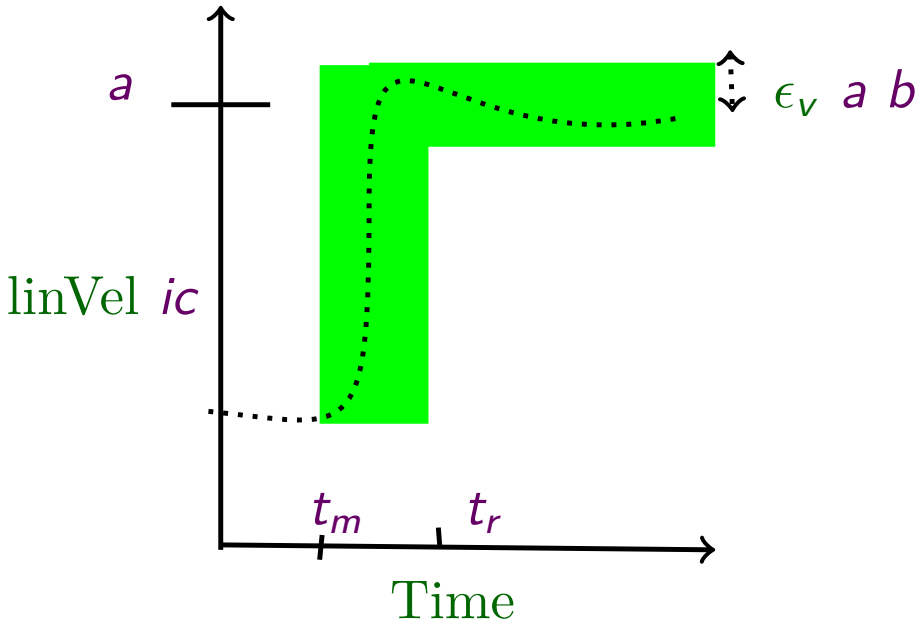
```
Definition HwAgent  (ic: iCreate) (evs : nat → option Event): Prop :=
onlyRecvEvts evs ∧ ∀ t: QTime,
  let (lastCmd, t_m ) := latestVelPayloadAndTime evs t in
  let a : Q := rad (lastCmd) in
  let b : Q := θ (lastCmd) in ∃ t_r : QTime, (t_m ≤ t_r ≤ t_m + reacTime)
    ∧ (∀ t' : QTime, (t_m ≤ t' ≤ t_r)
        → ( Min ({linVel ic} t_m) (a - ε_v a b)
            ≤ {linVel ic} t' ≤ Max ({linVel ic} t_m) (a+ ε_v a b)))
    ∧ (∀ t' : QTime, (t_r ≤ t' ≤ t) → |{linVel ic} t' - a | ≤ ε_v a b )
```



Specification of an iCreate (hardware agent)

⇑

`Definition` HwAgent (*ic*: iCreate) (*evs* : nat → option *Event*): `Prop` :=



{| v := *a*; w := *b* |} ⇑

Software Agents

$$S \to \text{Message} \to \quad (S \times \mathsf{list}\ \text{Message}). \qquad \Uparrow$$

Software Agents

$$S \to \text{Message} \to \quad (S \times \text{list Message}).$$

$$\text{SwSemantics} : \quad (S \to \text{Message} \to (S \times \text{list Message})) \to$$
$$(\mathit{PhysModelType} \to (\mathbb{N} \to \text{option } \mathit{Event}) \to \texttt{Prop})$$

Software Agents

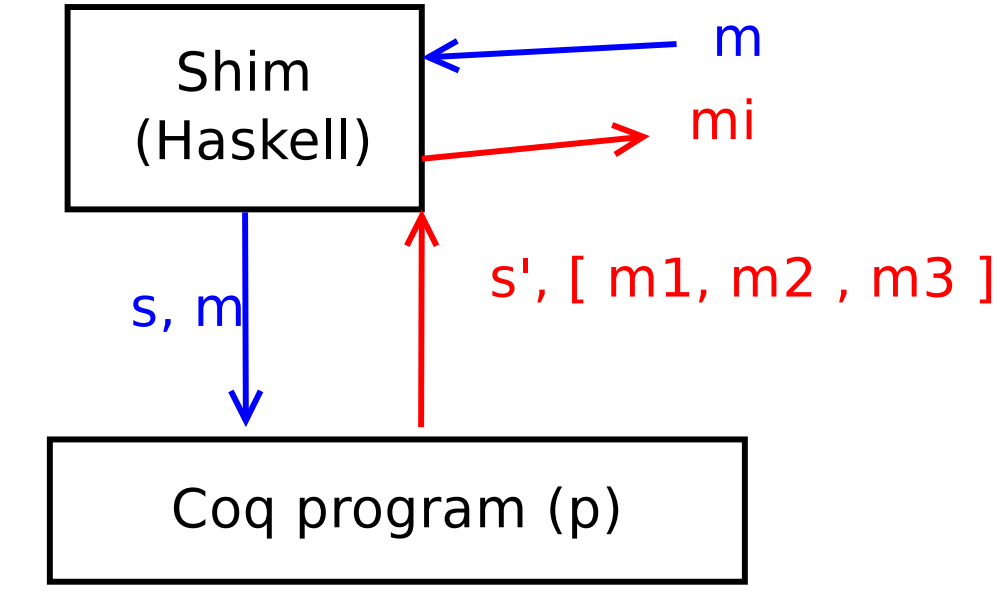
$$S \to \text{Message} \to \quad (S \times \text{list Message}).$$

$$\text{SwSemantics} : \quad (S \to \text{Message} \to (S \times \text{list Message})) \to$$
$$(\mathit{PhysModelType} \to (\mathbb{N} \to \text{option } \mathit{Event}) \to \texttt{Prop})$$

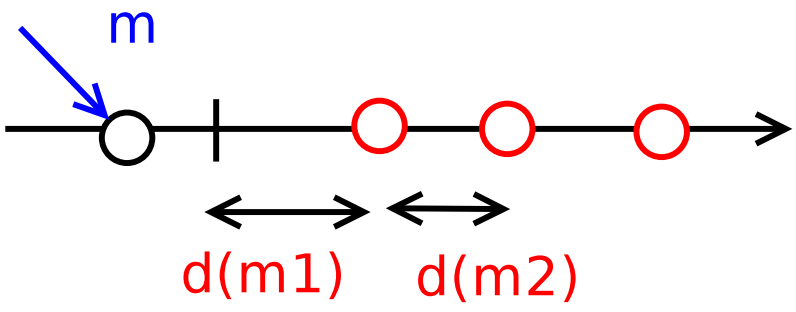
Software Agents

p: $S \to \text{Message} \to \quad (S \times \text{list Message})$.

$\text{SwSemantics} : \quad (S \to \text{Message} \to (S \times \text{list Message})) \to$
$(\mathit{PhysModelType} \to (\mathbb{N} \to \text{option } \mathit{Event}) \to \texttt{Prop})$



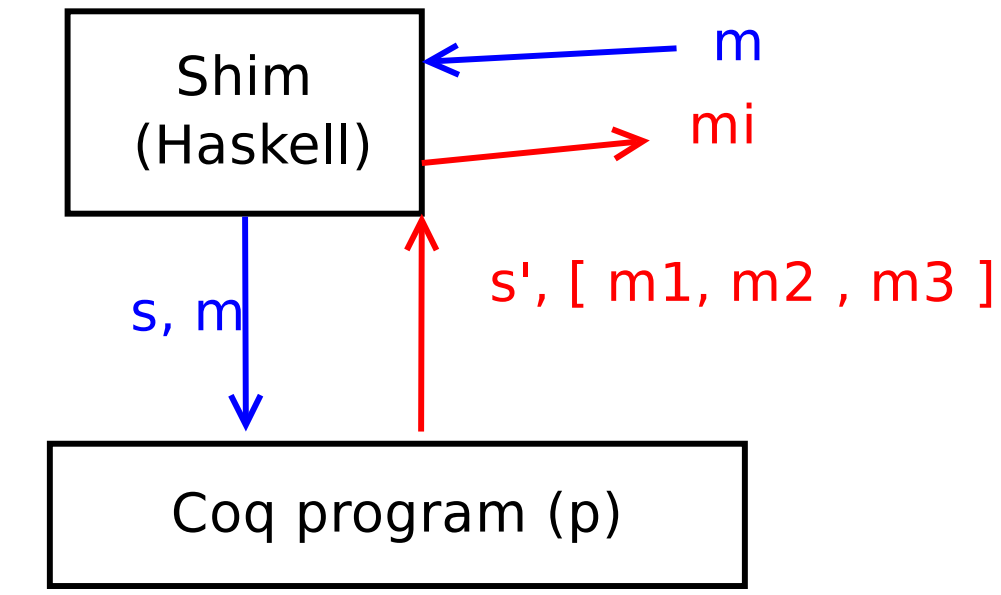
⇑

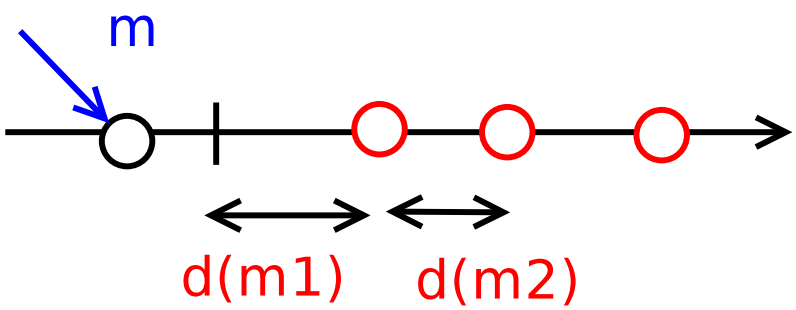


Software Agents

p: $S \to \text{Message} \to (S \times \text{list Message})$.

$\text{SwSemantics} : (S \to \text{Message} \to (S \times \text{list Message})) \to$
$(\mathit{PhysModelType} \to (\mathbb{N} \to \text{option } \mathit{Event}) \to \texttt{Prop})$

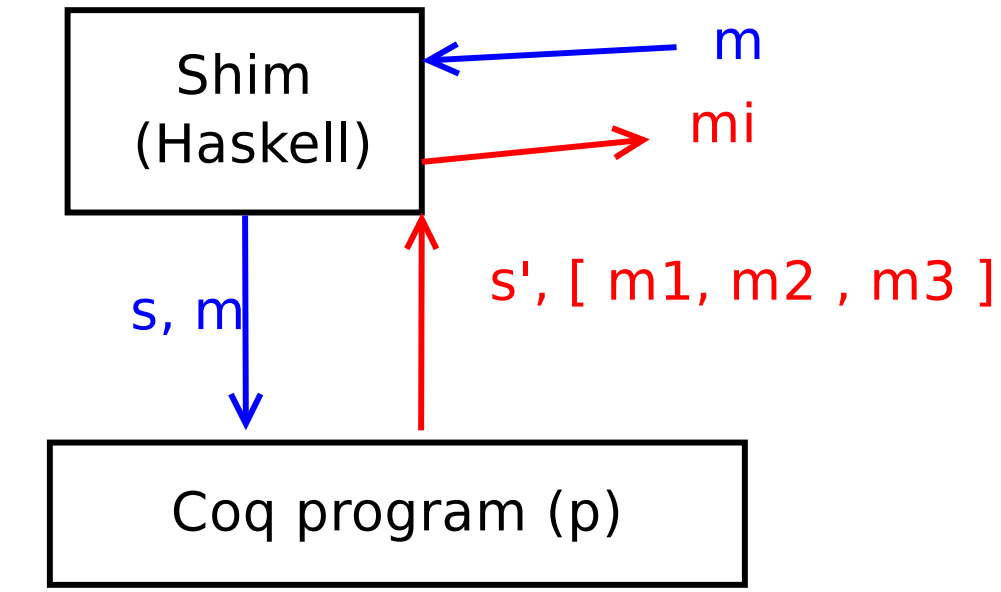


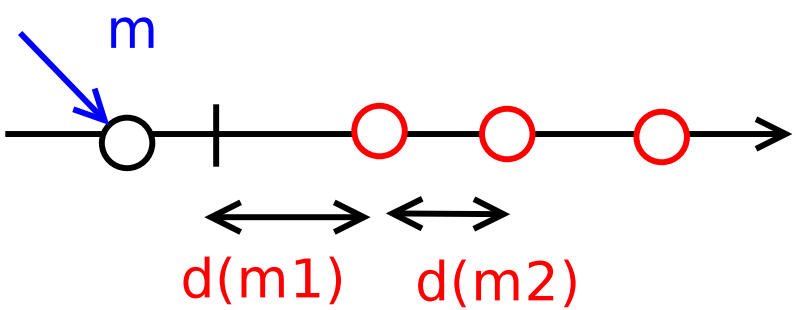


Software Agents

p: $S \to \text{Message} \to (S \times \text{list Message})$.

$\text{SwSemantics} : (S \to \text{Message} \to (S \times \text{list Message})) \to$
$(\mathit{PhysModelType} \to (\mathbb{N} \to \text{option } \mathit{Event}) \to \texttt{Prop})$





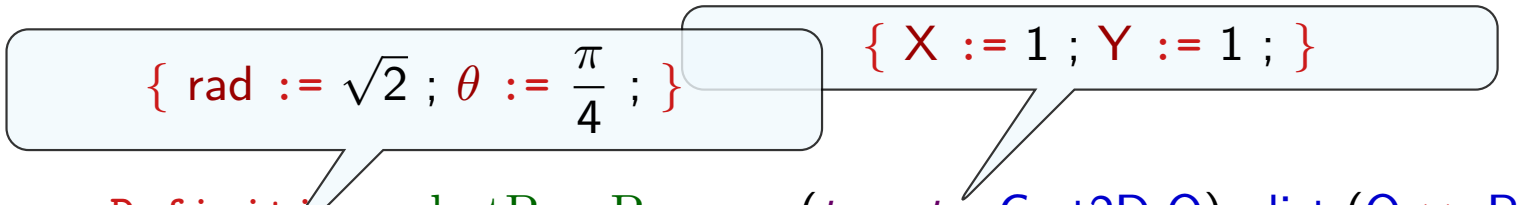
The Program in our running example

```
                                                                    ⇑
Definition robotPureProgam (target : Cart2D Q) : list (Q ×Polar2D Q) :=
  let polarTarget : Polar2D ℝ := Cart2Polar target in
  let rotDuration : ℝ := | θ polarTarget | / rotspeed in
  let translDuration : ℝ := (rad polarTarget) / speed in
  [

      (0,{| rad:= 0 ; θ := ( polarθSign target ) * rotspeed |})
      ; ( tapprox rotDuration delRes delEps , {| rad := 0 ; θ := 0 |})
      ; (delay , {| rad :=  speed ; θ := 0 |})
      ; ( tapprox translDuration res ϵ , {| rad := 0 ; θ := 0 |})  ].
```

The Program in our running example

```
Definition robotPureProgam (target : Cart2D Q) : list (Q × Polar2D Q) :=
  let polarTarget : Polar2D ℝ := Cart2Polar target in
  let rotDuration : ℝ := | θ polarTarget | / rotspeed in
  let translDuration : ℝ := (rad polarTarget) / speed in
  [

      (0,{| rad:= 0 ; θ := ( polarθSign target ) * rotspeed |})
      ; ( tapprox rotDuration delRes delEps , {| rad := 0 ; θ := 0 |})
      ; (delay , {| rad :=  speed ; θ := 0 |})
      ; ( tapprox translDuration res ϵ , {| rad := 0 ; θ := 0 |})  ].
```

The Program in our running example




```
Definition robotPureProgam (target : Cart2D Q) : list (Q × Polar2D Q) :=
  let polarTarget : Polar2D ℝ := Cart2Polar target in
  let rotDuration : ℝ := | θ polarTarget | / rotspeed in
  let translDuration : ℝ := (rad polarTarget) / speed in
  [

      (0,{| rad:= 0 ; θ := ( polarθSign target ) * rotspeed |})
      ; ( tapprox rotDuration delRes delEps , {| rad := 0 ; θ := 0 |})
      ; (delay , {| rad :=  speed ; θ := 0 |})
      ; ( tapprox translDuration res ϵ , {| rad := 0 ; θ := 0 |})  ].
```

The Program in our running example

$\frac{\pi}{4}$

```
Definition robotPureProgam (target : Cart2D Q) : list (Q × Polar2D Q) :=
  let polarTarget : Polar2D ℝ := Cart2Polar target in
  let rotDuration : ℝ := | θ polarTarget | / rotspeed in
  let translDuration : ℝ := (rad polarTarget) / speed in
  [

    (0,{| rad:= 0 ; θ := ( polarθSign target ) * rotspeed |})
    ; ( tapprox rotDuration delRes delEps , {| rad := 0 ; θ := 0 |})
    ; (delay , {| rad :=  speed ; θ := 0 |})
    ; ( tapprox translDuration res ϵ , {| rad := 0 ; θ := 0 |})  ].
```

The Program in our running example

$\sqrt{2}$

```
Definition robotPureProgam (target : Cart2D Q) : list (Q × Polar2D Q) :=
let polarTarget : Polar2D ℝ := Cart2Polar target in
let rotDuration : ℝ := | θ polarTarget | / rotspeed in
let translDuration : ℝ := (rad polarTarget) / speed in
[

    (0,{| rad:= 0 ; θ := ( polarθSign target ) * rotspeed |})
    ; ( tapprox rotDuration delRes delEps , {| rad := 0 ; θ := 0 |})
    ; (delay , {| rad :=  speed ; θ := 0 |})
    ; ( tapprox translDuration res ϵ , {| rad := 0 ; θ := 0 |})  ].
```

The Program in our running example

```
Definition robotPureProgam (target : Cart2D Q) : list (Q × Polar2D Q) :=
  let polarTarget : Polar2D ℝ := Cart2Polar target in
  let rotDuration : ℝ := | θ polarTarget | / rotspeed in
  let translDuration : ℝ := (rad polarTarget) / speed in
  [

      (0,{| rad:= 0 ; θ := ( polarθSign target ) * rotspeed |})
      ; ( tapprox rotDuration delRes delEps , {| rad := 0 ; θ := 0 |})
      ; (delay , {| rad :=  speed ; θ := 0 |})
      ; ( tapprox translDuration res ϵ , {| rad := 0 ; θ := 0 |})  ].
```

The Program in our running example

```
Definition robotPureProgam (target : Cart2D Q) : list (Q × Polar2D Q) :=
  let polarTarget : Polar2D ℝ := Cart2Polar target in
  let rotDuration : ℝ := | θ polarTarget | / rotspeed in
  let translDuration : ℝ := (rad polarTarget) / speed in
  [

      (0,{| rad:= 0 ; θ := ( polarθSign target ) * rotspeed |})
      ; ( tapprox rotDuration delRes delEps , {| rad := 0 ; θ := 0 |})
      ; (delay , {| rad :=  speed ; θ := 0 |})
      ; ( tapprox translDuration res ϵ , {| rad := 0 ; θ := 0 |})  ].
```

The Program in our running example

```
Definition robotPureProgam (target : Cart2D Q) : list (Q × Polar2D Q) :=
  let polarTarget : Polar2D ℝ := Cart2Polar target in
  let rotDuration : ℝ := | θ polarTarget | / rotspeed in
  let translDuration : ℝ := (rad polarTarget) / speed in
  [

      (0,{| rad:= 0 ; θ := ( polarθSign target ) * rotspeed |})
      ; ( tapprox rotDuration delRes delEps , {| rad := 0 ; θ := 0 |})
      ; (delay , {| rad :=  speed ; θ := 0 |})
      ; ( tapprox translDuration res ϵ , {| rad := 0 ; θ := 0 |})  ].
```

The Program in our running example

```
Definition robotPureProgam (target : Cart2D Q) : list (Q × Polar2D Q) :=
  let polarTarget : Polar2D ℝ := Cart2Polar target in
  let rotDuration : ℝ := | θ polarTarget | / rotspeed in
  let translDuration : ℝ := (rad polarTarget) / speed in
  [

      (0,{| rad:= 0 ; θ := ( polarθSign target ) * rotspeed |})
      ; ( tapprox rotDuration delRes delEps , {| rad := 0 ; θ := 0 |})
      ; (delay , {| rad :=  speed ; θ := 0 |})
      ; ( tapprox translDuration res ϵ , {| rad := 0 ; θ := 0 |})  ].
```

The Program in our running example

```
Definition robotPureProgam (target : Cart2D Q) : list (Q × Polar2D Q) :=
  let polarTarget : Polar2D ℝ := Cart2Polar target in
  let rotDuration : ℝ := | θ polarTarget | / rotspeed in
  let translDuration : ℝ := (rad polarTarget) / speed in
  [

      (0,{| rad:= 0 ; θ := ( polarθSign target ) * rotspeed |})
      ; ( tapprox rotDuration delRes delEps , {| rad := 0 ; θ := 0 |})
      ; (delay , {| rad :=  speed ; θ := 0 |})
      ; ( tapprox translDuration res ϵ , {| rad := 0 ; θ := 0 |})  ].
```

tapprox *r 1000* ϵ is a rational of the form $\frac{\cdots}{1000}$

$|$ tapprox *r res* ϵ - r $| \leq \frac{1+2*\epsilon}{2*res}$

Robbert Krebbers and Bas Spitters. "Type classes for efficient exact real arithmetic in Coq". In: *LMCS* 9.1 (Feb. 14, 2013)

Constructive Reals

$\mathrm{CR} \cong \mathbb{Q}^+ \to \mathbb{Q}$

Robbert Krebbers and Bas Spitters. "Type classes for efficient exact real arithmetic in Coq". In: *LMCS* 9.1 (Feb. 14, 2013)

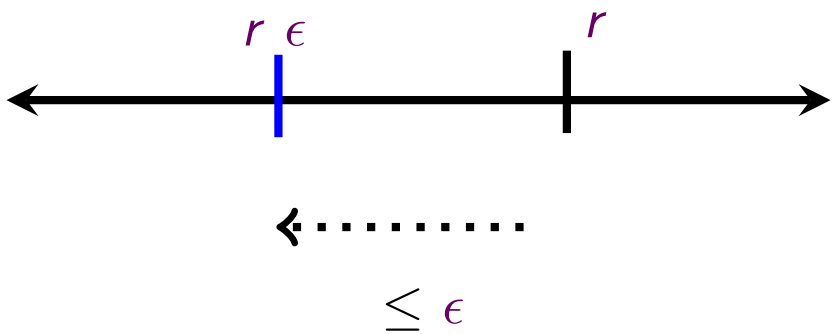
Errett Bishop. *Foundations of constructive analysis*. McGraw-Hill, 1967. 394 pp.

Constructive Reals

$\mathrm{CR} \cong \mathbb{Q}^+ \to \mathbb{Q}$

$r : \mathrm{CR}$
$\epsilon : \mathbb{Q}^+$

$r\ \epsilon : \mathbb{Q}$

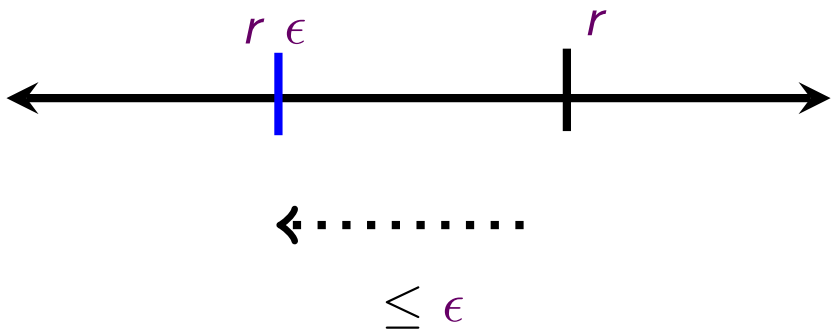


Constructive Reals

$\mathrm{CR} \cong \mathsf{Q}^+ \to \mathsf{Q}$

$r : \mathrm{CR}$
$\epsilon : \mathsf{Q}^+$

$r\ \epsilon : \mathsf{Q}$



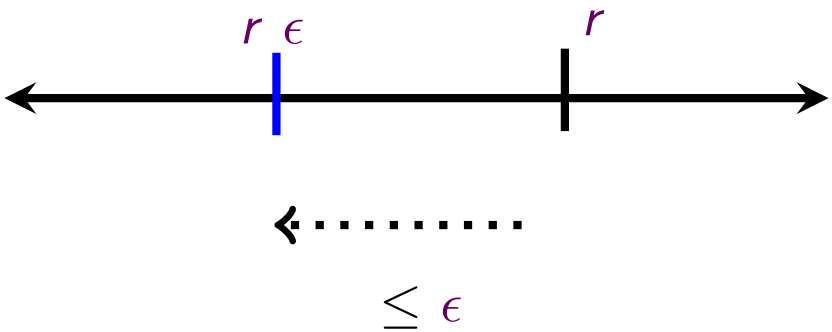
$+ : \lambda\ (r_1 : \mathrm{CR})\ (r_2 : \mathrm{CR}).$

Constructive Reals

$\mathrm{CR} \cong \mathbb{Q}^+ \to \mathbb{Q}$

$r : \mathrm{CR}$
$\epsilon : \mathbb{Q}^+$

$r\ \epsilon : \mathbb{Q}$



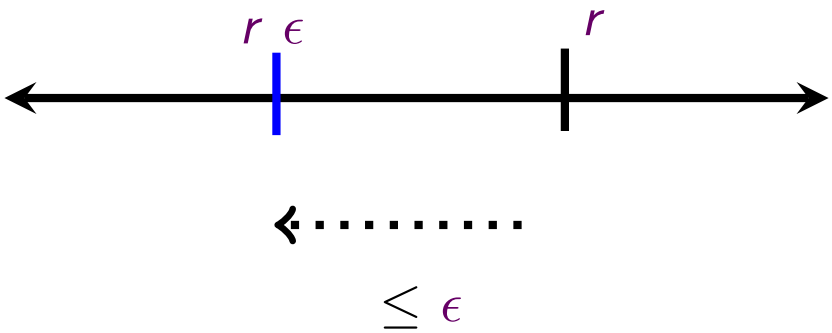
$+ : \lambda\ (r_1 : \mathrm{CR})\ (r_2 : \mathrm{CR}).\ \ \lambda\ (\epsilon : \mathbb{Q}^+)\ .$

Constructive Reals

$\mathrm{CR} \cong \mathbb{Q}^+ \to \mathbb{Q}$

$r : \mathrm{CR}$
$\epsilon : \mathbb{Q}^+$

$r\ \epsilon : \mathbb{Q}$



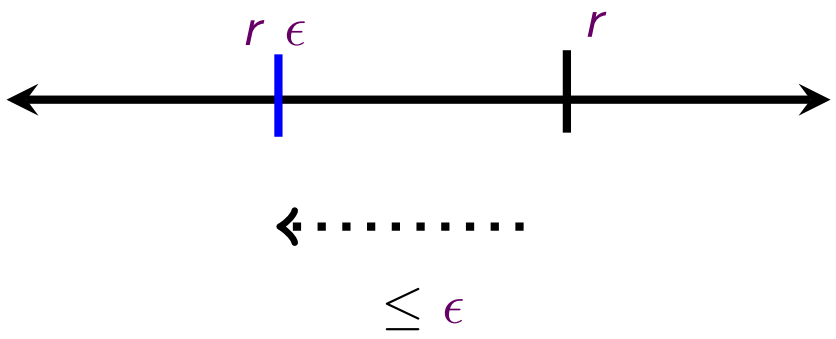
$+ : \lambda\ (r_1 : \mathrm{CR})\ (r_2 : \mathrm{CR}).\ \ \lambda\ (\epsilon : \mathbb{Q}^+)\ .\ \ \ (r_1\ \frac{\epsilon}{2} + r_1\ \frac{\epsilon}{2})$

Constructive Reals

$$\mathrm{CR} \cong \mathbb{Q}^+ \to \mathbb{Q}$$

$$r : \mathrm{CR}$$
$$\epsilon : \mathbb{Q}^+$$

$$r\ \epsilon : \mathbb{Q}$$



$$+ : \lambda\ (r_1 : \mathrm{CR})\ (r_2 : \mathrm{CR}).\ \ \lambda\ (\epsilon : \mathbb{Q}^+)\ .\ \ \ (r_1\ \tfrac{\epsilon}{2} + r_1\ \tfrac{\epsilon}{2})$$

$$\forall\ (\theta : \mathrm{CR}).\ (\cos\ \theta)^2 + (\sin\ \theta)^2 = 1$$

Discrete computations over CR

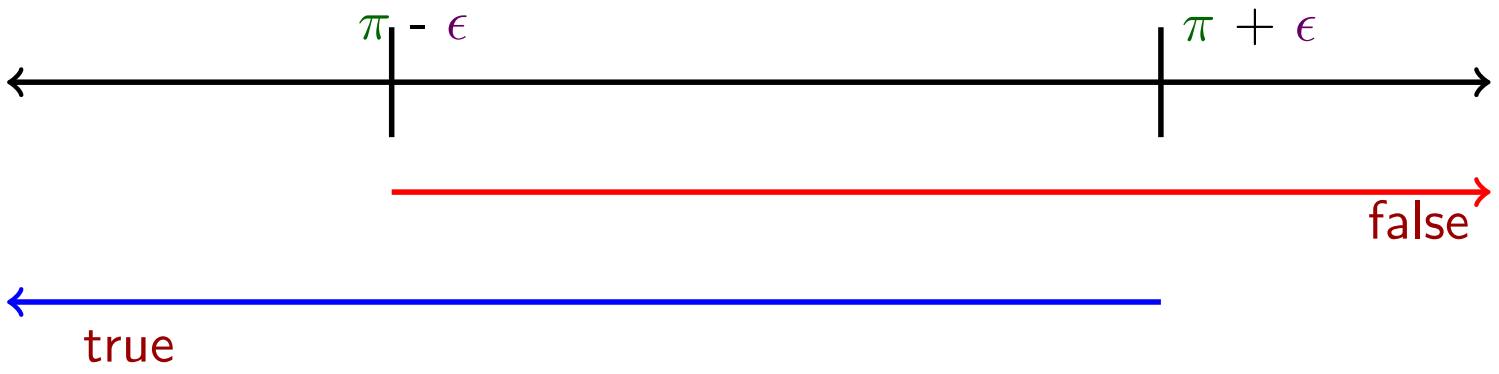
$$\texttt{if}\ (r < \pi)\ \text{then}\ \ldots\ \text{else}\ \ldots$$

te computations over CR

$$\texttt{if}\ (r < \pi)\ \ldots\ \text{else}\ \ldots$$

Discrete computations over CR

$$\texttt{if}\ (r <_\epsilon \pi)\ \text{then}\ \ldots\ \text{else}\ \ldots$$

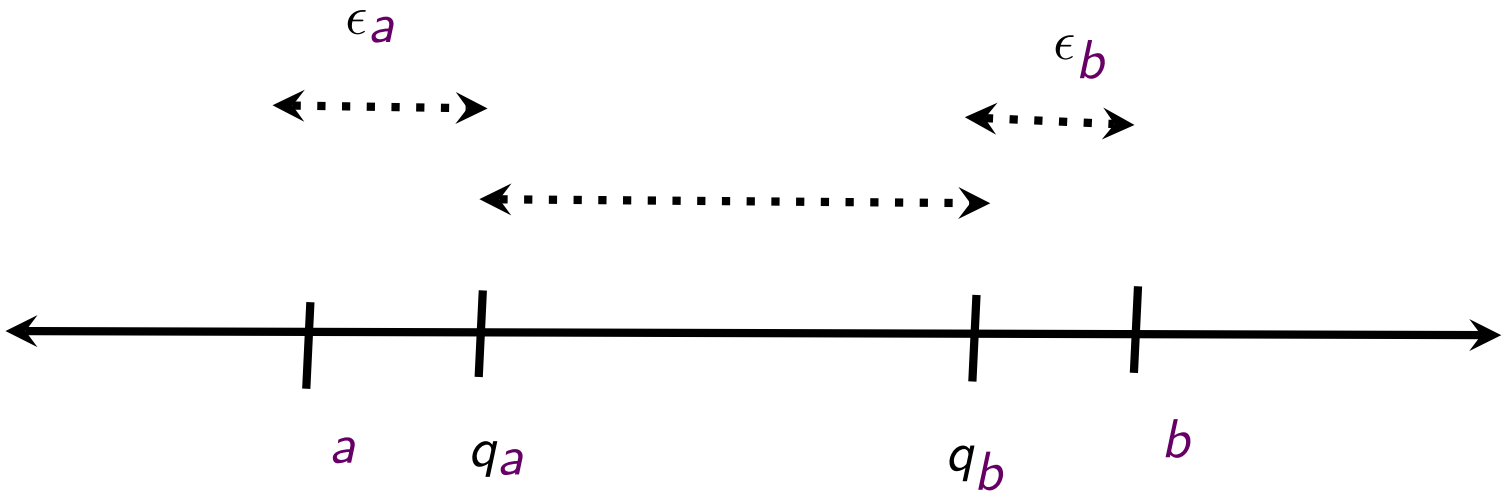


Constructivity helps even in logical reasoning

$$\sqrt{((cos\ \tfrac{1}{2}))} < exp(cos(sin(arctan\ \pi)))$$

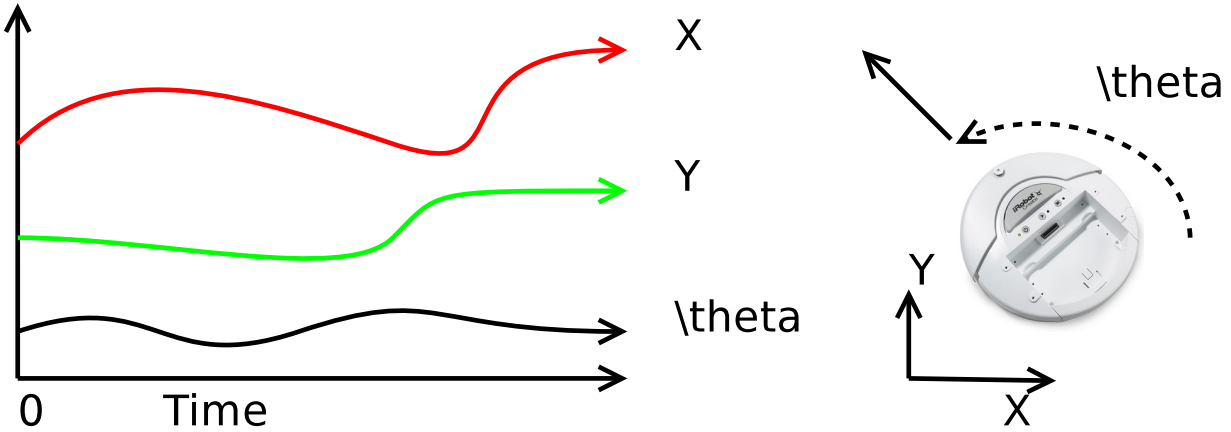
Constructivity helps even in logical reasoning

$$\sqrt{((cos\ \tfrac{1}{2}))} < exp(cos(sin(arctan\ \pi)))$$



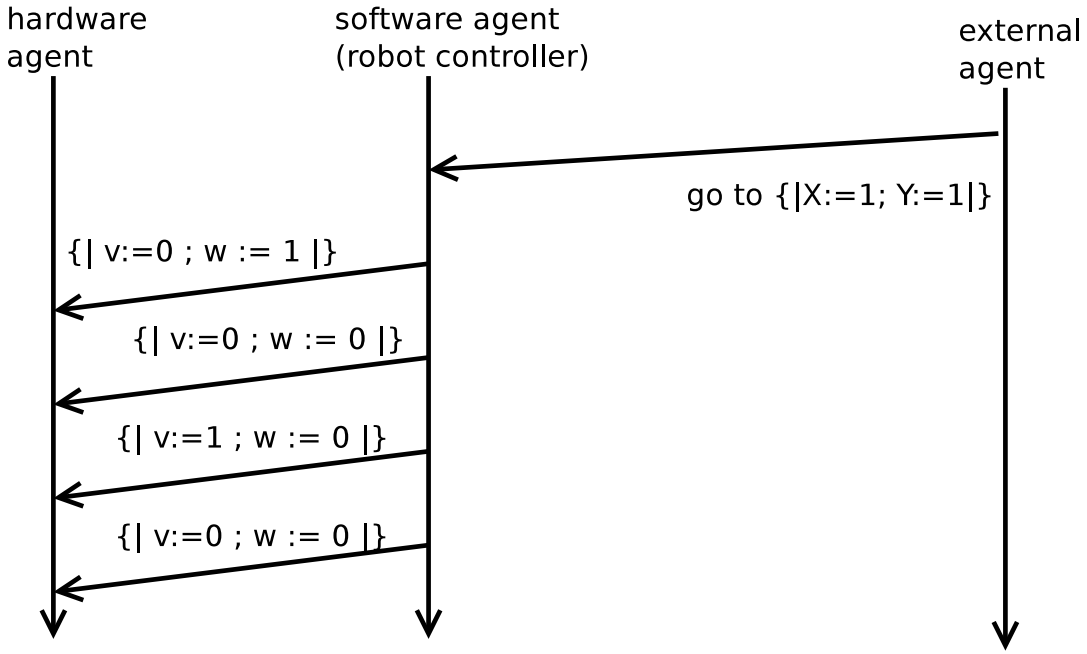
Recap: How to Specify a CPS in ROSCoq

- Define the physical model as a Coq type

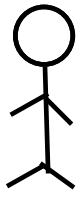


Recap: How to Specify a CPS in ROSCoq

- Define the physical model as a Coq type
- Define the collection of agents

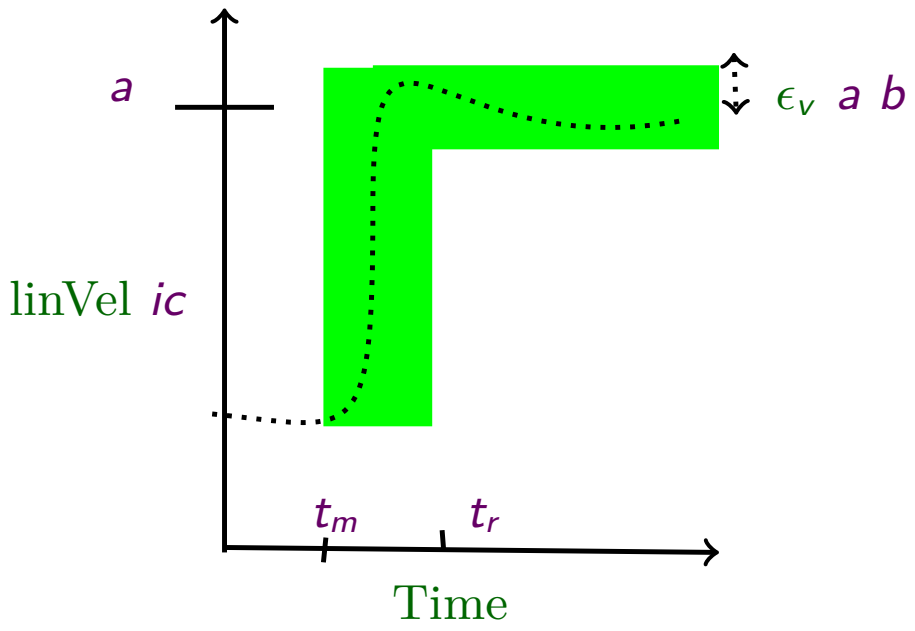


```
Definition robotPureProgam (target : Cart2D Q) : list (Q × Polar2D Q) :=
  let polarTarget : Polar2D ℝ := Cart2Polar target in
  let rotDuration : ℝ := | θ polarTarget | / rotspeed in
  let translDuration : ℝ := (rad polarTarget) / speed in
  [ (0,{| rad:= 0 ; θ := ( polarθSign target ) * rotspeed |})
      ; ( tapprox rotDuration delRes delEps , {| rad := 0 ; θ := 0 |})
      ; (delay , {| rad :=  speed ; θ := 0 |})
      ; ( tapprox translDuration delRes delEps , {| rad := 0 ; θ := 0 |}) ].
```



Recap: How to Specify a CPS in ROSCoq

- Define the physical model as a Coq type
- Define the collection of agents
- Specify the behavior of each agent
  - H/w agent : Coq relation

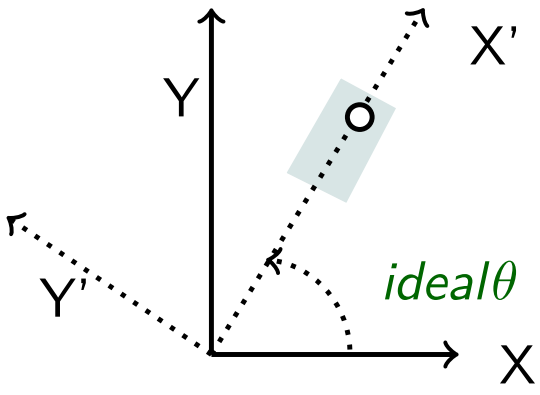


Recap: How to Specify a CPS in ROSCoq

- Define the physical model as a Coq type
- Define the collection of agents
- Specify the behavior of each agent
  - H/w agent : Coq relation
  - S/w agent : Coq message handler

```
Definition robotPureProgam (target : Cart2D Q) : list (Q × Polar2D Q) :=
  let polarTarget : Polar2D ℝ := Cart2Polar target in
  let rotDuration : ℝ := | θ polarTarget | / rotspeed in
  let translDuration : ℝ := (rad polarTarget) / speed in
  [
      (0,{| rad:= 0 ; θ := ( polarθSign target ) * rotspeed |})
      ; ( tapprox rotDuration delRes delEps , {| rad := 0 ; θ := 0 |})
      ; (delay , {| rad :=  speed ; θ := 0 |})
      ; ( tapprox translDuration res ϵ , {| rad := 0 ; θ := 0 |}) ].
```

Proved Property



⇑

```
Definition ErrY': ℝ := (ϵv 0 w) * ( reacTime + Ev01TimeGapUB)
+ (Sin (θErrTrans + θErrTurn)) * ( | target | + speed*timeErr
                                    + Ev23TimeGapUB * (ϵv speed 0) )
.
```

Experiments

target		actual		video link
X	Y	X	Y	
-1	1	-1.06	0.94	vid1
-1	-1	-1.02	-0.99	vid2
1	1	1.05	0.94	vid3

Ongoing/Future Work

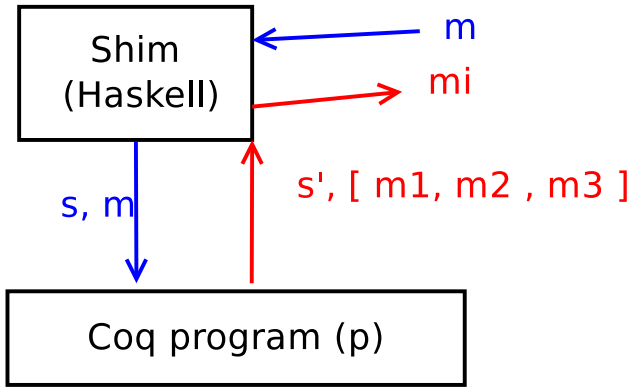
- General Shim in Haskell[6][7]
  - Importing ROS message types in Coq

[6] https://github.com/acowley/roshask

[7] Anthony Cowley and Camillo J. Taylor. "Stream-oriented robotics programming: The design of roshask". In: *IROS*. IEEE, 2011, pp. 1048–1054.

Ongoing/Future Work

- General Shim in Haskell[6][7]
  - Importing ROS message types in Coq
  - Writing the main loop

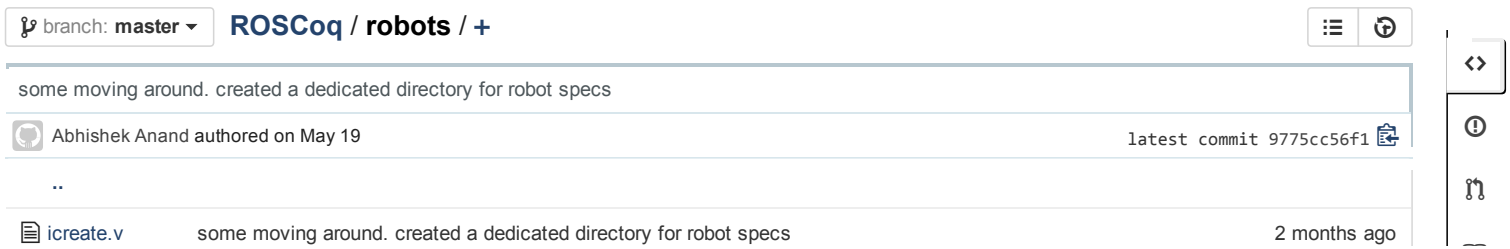


[6] https://github.com/acowley/roshask

[7] Anthony Cowley and Camillo J. Taylor. "Stream-oriented robotics programming: The design of roshask". In: *IROS*. IEEE, 2011, pp. 1048–1054.

Ongoing/Future Work

- General Shim in Haskell[6][7]
  - Importing ROS message types in Coq
  - Writing the main loop
- Library of formally specified robots



[6] https://github.com/acowley/roshask

[7] Anthony Cowley and Camillo J. Taylor. "Stream-oriented robotics programming: The design of roshask". In: *IROS*. IEEE, 2011, pp. 1048–1054.

Ongoing/Future Work

- General Shim in Haskell[6][7]
  - Importing ROS message types in Coq
  - Writing the main loop
- Library of formally specified robots

[6] https://github.com/acowley/roshask

[7] Anthony Cowley and Camillo J. Taylor. "Stream-oriented robotics programming: The design of roshask". In: *IROS*. IEEE, 2011, pp. 1048–1054.

Ongoing/Future Work

- General Shim in Haskell[6][7]
  - Importing ROS message types in Coq
  - Writing the main loop
- Library of formally specified robots
- Verifying multi-robot collaboration

[6] https://github.com/acowley/roshask

[7] Anthony Cowley and Camillo J. Taylor. "Stream-oriented robotics programming: The design of roshask". In: *IROS*. IEEE, 2011, pp. 1048–1054.

Ongoing/Future Work

- General Shim in Haskell[6][7]
  - Importing ROS message types in Coq
  - Writing the main loop
- Library of formally specified robots
- Verifying multi-robot collaboration
- Probabilistic reasoning[8]

---

[6] https://github.com/acowley/roshask

[7] Anthony Cowley and Camillo J. Taylor. "Stream-oriented robotics programming: The design of roshask". In: *IROS*. IEEE, 2011, pp. 1048–1054.

[8] Thierry Coquand and Erik Palmgren. "Metric Boolean algebras and constructive measure theory". In: *Archive for Mathematical Logic* 41.7 (Oct. 1, 2002), pp. 687–704.

Questions

http://www.cs.cornell.edu/~aa755/ROSCoq/
http://github.com/aa755/ROSCoq