Dependent types, 2-categories and Cyber-Physical System Applications

Georgios Bakirtzis Jacques Carette

November 2, 2020

1 Types

Here I will try to develop some types. Will probably need input and back and forth with Carette.

A cyber-physical system with an associated phase/state space could have the following type:

Type := CPS [states: Set; actions: Set; network: Grph;]

Therefore, the networked aspect is Grph-labelled while the states and actions are Set-labelled. This and the hereafter decompositions (which will be discussed relate to Carette [1].

Will probably then say how this relates to: sensors, controller, actuators in terms of Drasil [**?**]. What I mean by this is that the physical quantities that are produced in the "physical" portion of cyber-physical will be modeled in terms of Drasil, which the "cyber" parts will be modeled based on dependent types. does that sound good?

2 Notes

These are the three papers we talked about [3, 1, 2].

In relation to CPS we talked about vertical decomposition in the form of types.

From the Lawvere understanding of a functor translating syntax to semantics to a 2-categorical perspective: one category models syntax while another models semantics.

This allows us to completely decouple how we name things – which is important in system design – and what the naming corresponds to.

We talked specifically about how a sensor in a control system can be one of many things even in the sense of behavior.

My general idea is to talk about decomposition the same way Carette talks about mathematical constructs (e.g., magma $\rightarrow$ abelian group) and also for the physical side talk about units as they are presented in Drasil.

I think merging those two idea for "cyber-physical" modeling would be a good first paper.

Will we address operads in terms of how this is different (we definitely want to address monoidal categories + extra structure)

3 Make the UAV types follow Carette

The following is very rough.

From the controls point of view there is often no external input to the sensor. This is because sensor is this context measures the system. However, while this is true almost always it is not

complete with respect to the implementation of the system. The sensors play a congruent but important role, they take measurements of both the system and the environment, thereby allowing the plane to actually fly even without an internal model of the system within the controller. From a systems and software engineering point of view we want to model both the estimated state produced by the sensor and the translation of the environment to distinct physical measurements to digital signals. We could envision the type signature for the sensor as taking in the environment and state and producing a new state,

$$L : e, s \to s'.$$

The environment e for the UAV application is a measurement of physical quantities corresponding to position, attitude or otherwise orientation, and speed. The states, s, s' represent the attitude and the speed at a particular point in time.

UAV systems are often controlled using proportional–integral–derivative (PID) controller by sending an electric signal to small motors, called servos, which in turn control the position of the plane by manipulating the aileron, rudder, elevator, and throttle. In practice, an external signal either through the ground control station or an actual pilot is given to describe the desireable flight path. In UAVs this is a set of waypoints that is sent outside the UAV.

The job of the controller is to take this desired state and apply the correct actions to fly the plane from waypoint A to waypoint B and beyond. An additional predicted state is used for error correction. We can model the controller through the type signature

$$C : (s', d) \to \mathrm{c}$$

taking in a predicted state s', a desired state d, and producing some control action through servos c.

By controlling the different control surfaces servos ultimate control the plane's *attitude* and correspondingly the pitch, roll, and yaw through

$$D : c \to \mathrm{s}.$$

Dynamics does a lot more in terms of airframe, position, attitude, etc. However, at this level of abstraction it is useful to think about dynamics as taking some input from the electromechanical subsystems, which is captured in the motors that control the plane, and produces a new state. This new state in the case of a UAV takes the form of movement of the plane from one waypoint to another. The compositional approach to CPS design is flexible in terms of further decompositions, meaning it is possible to decompose the mechanical elements of the UAV, for example, the airframe can be decomposed to wings, wheels, propeller, et cetera.

4 Some more type stuff lifted from word graveyard

Given a linear time-invariant system which inhabits some $\mathbb{R} \to \boxed{A} \to \mathbb{R}$ and is completely characterized by its set of states S and matrices A, B, C, we can always first of all make the structural choice to 'decompose' the surrounding interface as in **??** using the wiring diagram $(f_{\text{in}}, f_{\text{out}})\colon L \otimes C \otimes D \to A$, and subsequently determine an object in the pre-image of $(S, A, B, C) \in FA$ under the functor $F(f_{\text{in}}, f_{\text{out}})$. This object in the pre-image turns out to actually assign the very same matrices A, B, C to the box D (which is reasonable, being the dynamics? I know you only wanted

A to be shared or something) and then leave some flexibility on what we assign on S and C boxes. (we should elaborate on that, by giving at least an example of what specifically S and C can be!)

A possible set of components for the implementation of the sensory system for a flight control system includes a gyroscope, an accelerometer, and a pressure sensor, which are usually realized through an inertial measurement unit (IMU). Because an IMU has accumulative error it is important to either add redundancy; that is, use two IMU devices or add a separate device, for example, a global positioning unit (GPS) to ping and correct this error in IMU measurements. Current UAV systems include further functionality, for example, by using differential pressure sensors or adding a magnetometer to achieve further robustness during guided flight. Here we will focus on one possible design of the UAV, including implementation for the dynamics through control surfaces and airframe.

From the controls point of view there is often no input to the sensor. This is because sensor is this context measures the system. However, while this is true almost always it is not complete with respect to the implementation of the system. The sensors play a congruent but important role, they take measurements of both the system and the environment, thereby allowing the plane to actually fly even without an internal model of the system within the controller. From a systems and software engineering point of view we want to model both the estimated state produced by the sensor and the translation of the environment to distinct physical measurements to digital signals. We could envision the type signature for the sensor as taking in the environment and state and producing a new state, $\mathtt{L} : e, s \to s'$. The environment e for the UAV application is a measurement of physical quantities corresponding to position, attitude or otherwise orientation, and speed. The states, s, s' represent the attitude and the speed at a particular point in time. The choices of what semantics to apply to the wires is also a design decision, in this case we have the minimum amount of information to fly a fixed-winged plane.

UAV systems are often controlled using proportional–integral–derivative (PID) controller by sending an electric signal to small motors, called servos, which in turn control the position of the plane by manipulating the aileron, rudder, elevator, and throttle. In practice, an external signal either though the ground control station or an actual pilot is given to describe the desireable flight path. In UAVs this is a set of waypoints that is sent outside the UAV. The job of the controller is to take this desired state and apply the correct actions to fly the plane from waypoint A to waypoint B and beyond. An additional predicted state is used for error correction. We can model the controller through the type signature $\mathtt{C} : (s', d) \to \mathrm{c} \to$ servos.

By controlling the different control surfaces servos ultimate control the plane's *attitude* and correspondingly the pitch, roll, and yaw through $\mathtt{D} :$ servos $\to$ s.

Dynamics does a lot more in terms of airframe, position, attitude, etc. However, at this level of abstraction it is useful to think about dynamics as taking some input from the electromechanical subsystems, which is captured in the motors that control the plane, and produces a new state. This new state in the case of a UAV takes the form of movement of the plane from one waypoint to another. The compositional approach to CPS design is flexible in terms of further decompositions, meaning it is possible to decompose the mechanical elements of the UAV, for example, the airframe can be decomposed to wings, wheels, propeller, et cetera.

We could further decompose. A useful example might be the processing unit of the flight control system, which has the following type signature

$$\mathtt{P1} : \text{orientation, speed} \to (\text{state}).$$

From the point of view of this simplified system

$$I : attitude$$

P2:

The interesting thing about these type signatures is that even though they where derived hierarchically from the physics model of the UAV, they correspond – as might be expected – to the software implementation of the flight control system. These type variables can be used in a number of ways, two of which include decomposing to a software implementation from the modeling language itself and type checking the information in the eventual deployment of the system. These type signatures can be augmented by the contracts algebra. In the implementation of embedded software this could assure that certain conditions are not just type checked for correctness but are also constraint in some way by contracts, which in turn represent requirements. The vast majority of flight control systems today are software controlled and, therefore, the topic of constraining software implementations in a top-to-bottom fashion is increasingly pertinent.

References

[1] J. Carette and R. O'Connor. Theory presentation combinators. In *International Conference on Intelligent Computer Mathematics*. Springer, 2012.

[2] D. R. Smith. Mechanizing the development of software. *NATO ASI Series F Computer and Systems Sciences*, 1999.

[3] D. Szymczak, S. Smith, and J. Carette. Position paper: A knowledge-based approach to scientific software development. In *Proceedings of the 2016 IEEE/ACM International Workshop on Software Engineering for Science (SE4Science)*. IEEE, 2016.