Kalpa Publications in Computing

Volume 20, 2024, Pages 17–24

Proceedings of 2024 Concurrent Processes Architectures and Embedded Systems Hybrid Virtual Conference

Modelling a Guardrail for an AI Control System Using CSP

Jeremy M. R. Martin

Lloyd's of London

Abstract

I consider the problem of attempting to constrain the behaviour of artificial intelligence software to prevent it from carrying out unwanted, dangerous, or malicious operations. I shall analyse a medical appliance scenario: the artificial pancreas. I shall model a diabetes patient as a system which contains a certain level of insulin and a certain level of glucose at any given point in time, and may be engaged in eating, rest, or exercise. Sugar levels will be lowered using up some insulin. Exercise will also contribute to lowering sugar levels. The artificial pancreas will periodically inject some more insulin into the patient depending on decisions made by an AI-driven control system when it is provided with monitoring data and it is intended to learn over time how best to manage the patient's glucose levels. There is also a safety system which may decide to suspend the AI system and trigger a manual or algorithmic override should a particular threshold be breached. I shall use the CSP process algebra to represent the processes and the associated FDR proof tool to deduce properties about their behaviour.

Keywords - CSP, Concurrency, AI.

1 Introduction

We are living in a time when artificial intelligence is almost never out of the news and the field is coming of age and penetrating many mainstream aspects of our society. The theoretical origins of AI go back to the days of Alan Turing, but we have only recently arrived at a situation where the computing power available to end users has crossed a threshold which makes AI readily available on a large scale. Possibilities of AI that were once in the realm of science fiction are becoming commonplace. We already have driverless cars on the streets in California. AI is being used for medicinal purposes to diagnose and treat illnesses. It is being used in creation of documentation and programming code. It proliferates in the worlds of music and art. And it is increasingly used for both cyber security and cyber terrorism. There are both significant opportunities and threats to humans within the realm of artificial intelligence. The latter concern has led to the UK government recently setting up the AI Safety Institute to "minimise surprise to the UK and humanity from rapid and unexpected advances in AI" [1].

In this paper I shall consider the specific problem of building guardrails to prevent AI control systems from running amok and making dangerous or malicious decisions with potentially devastating impact [2]. I have considered three areas to help illustrate this problem.

The first of these is the automated construction and pricing of insurance policies: a role that is typically carried out by actuaries and underwriters in the Lloyd's Insurance Market. It is in the nature of AI systems that they are trained in specific domains using relevant training sets of data, and then adapt their behaviours as they learn from their experience. They may perform tasks representative of human activities proficiently in those areas where they have been trained but doubts persist as to what they might do should they be plunged into a context which has not been explored as part of their training. The insurance industry is heavily regulated, and this regulation has largely come due to failings in the past where insufficient funds might have been put aside to payout to policy holders in the event of them experiencing major losses due to large volumes of claims occurring due to unforeseen events. In the case of the Lloyd's Insurance Market there was a major crisis in the 1980s when asbestos-related losses surfaced, covered by historical policies, which were devastating to investors. It is hard to see how an AI pricing engine could have reliably foreseen the possible appearance of a previously unknown health condition that took decades to emerge after the policies were priced. It would be interesting to consider how guardrail software could be used to warn against possible issues arising from decisions made by AI in this context.

A second problem that interests me is the safety of driverless cars. As a driver I have occasionally encountered very difficult driving conditions which I had never experienced before. For instance, I have been caught driving on a narrow coastal road in thick fog with very little visibility. I have also found myself driving on an icy country lane with many potholes in a blizzard. So, I wonder what would happen to an AI self-learning system, driving a car if it were suddenly to encounter a situation that had not been seen before in any of the training data? If I were travelling in such a vehicle, I would want some assurance of safety perhaps provided by an early warning system and the possibility of manual override.

Both above scenarios are of considerable interest to me, however I have decided to select the phenomenon of AI-driven medical appliances for a slightly deeper dive. These have the potential to save and extend lives but also carry risks of doing immeasurable harm to the individuals who depend upon them. I shall be looking at the artificial pancreas concept and how this might be prevented from accidentally causing harm by using a guardrail process. I am using the CSP and FDR technologies to analyse this space. Below, I shall describe how to model a highly simplified version of an interesting prototype system, called PEPPER [3] and then how these tools provide insights into its behaviour.

2 Case Study

The PEPPER system [3] was not in itself an artificial pancreas, it was more in the nature of an AI-informed decision support system which ran on a mobile phone. It was connected to certain monitoring apparatuses and would make recommendations to a person with diabetes as to how much insulin to ingest. It acted as a prototype for a real artificial pancreas that could be of huge benefit to those who suffer from type 1 diabetes.

PEPPER was developed by a consortium of scientists from Oxford Brookes University Imperial College London and the Institut d'Investigació Biomèdica de Girona. The system was tested on a group of adult volunteers and was found to improve the management of their condition.

Each patient using this system had the glucose level in their blood continually monitored by a wearable device, an insulin pump to supply multiple daily injections of insulin directly into their blood, an activity monitor which measures when they are exercising or at rest, and they were also required to enter information about all the food they consumed manually through the graphical user interface of the mobile application that ran on their phones. The mobile application included two AI tools: one was the Bolus Recommender which made recommendations to the patient about how much insulin to inject using the pump, and the other was the safety system which would predict the likelihood of impending adverse events: hypoglycaemia or hyperglycaemia, allowing patients to make informed dosing decisions.

The study concluded the following. “A safety system for an insulin dose recommender has been proven to be a viable solution to reduce the number of adverse events associated to glucose control in type 1 diabetes.”

3 Analysis using CSP

The CSP language of CAR Hoare (Communicating Sequential Processes) [4, 5, 6, 7, 8, 9] is a notation for describing patterns of communication by algebraic expressions. It is widely used for design of parallel and distributed hardware and software and for the formal proof of vital properties of such systems. (However, without computer assistance it is often impractical to prove such properties, algebraically, other than for very simple systems.) The grammar of CSP is shown in Figure 1

Process	:==	$STOP$	Deadlock
	|	$SKIP$	Termination
	|	$CHAOS$	Might do anything
	|	event $\rightarrow$ Process	Event prefix
	|	channel?x $\rightarrow$ Process	Input
	|	channel!x $\rightarrow$ Process	Output
	|	Process_1; Process_2	Sequential composition
	|	Process_1 $[\![\text{alph}_1 \mid \text{alph}_2]\!]$ Process_2	Parallel Composition
	|	$\text{Process}_1 \sqcap \text{Process}_2$	Non-deterministic choice
	|	$\text{Process}_1 \,\Box\, \text{Process}_2$	Deterministic choice
	|	**if** B **then** Process_1 **else** Process_2	Conditional
	|	Process $\setminus$ event	Event hiding
	|	f(Process)	Event relabelling
	|	name	

Figure 1. Grammar of the core CSP Language

The FDR tool (Failures Divergences Refinement) [8] provides an automated proof system for the CSP language when it is expressed in a machine readable format. In the following section we shall construct a model of an artificial pancreas system and provide a safety mechanism which runs alongside the AI control system. We shall then use FDR to reason about the behaviour of this system and how it might help to prevent adverse events for the patients.

An individual’s blood glucose level is affected by multiple factors, including carbohydrate consumption, physical activity, ambient temperature, illness, stress, sleep and insulin on board. We consider only carbohydrates and exercise for the purposes of this model.

We start by defining the CSP code to describe the system in question. We define some communication channels and their types. The channel ‘measure’ is used to connect the patient to the monitoring device and is used to return three measured values. These values are the level of glucose in the blood, whether or not eating has taken place since the last observation, and whether or not exercise has taken place since the last observation. We also define channels to represent the acts of eating and exercising by the patient, the decision by either the AI engine or the safety algorithm to inject some insulin.

```
channel measure, signal_ai, signal_alg: {0..9}.{0,1}.{0,1}
channel eat, exercise, inject_ai, inject_alg, inject: {0,1}
channel tock, alert
```

We define utility functions to calculate maximum and minimum of two numbers as follows.

```
max(a,b) = if a>b then a else b
min(a,b) = if a>b then b else a
```

Now let us define a process to represent a patient. This is simplified – we use abstract units to quantify insulin, time, food consumption and energy. It would be vital that this would be made into a far more realistic model if this approach were to be used for real. So, a patient is modelled as a process which maintains certain levels of glucose and insulin and at specified points in time may eat, exercise, rest or receive an injection of insulin. The ‘tock’ event represents the cadence of the monitoring intervals.

```
PATIENT(glucose, insulin) = tock -> eat?food -> exercise?activity ->
  if (insulin > 0 and glucose > 0 and food == 1) then
    measure!glucose.1.activity -> inject?dose ->
      PATIENT(glucose, insulin-1+dose)
  else if (insulin == 0 and glucose < 9 and food == 1) then
    measure!(glucose+1).1.activity -> inject?dose ->
      PATIENT(glucose+1,dose)
  else if (glucose > 0 and activity == 1) then
    measure!(glucose-1).food.1 -> inject?dose ->
      PATIENT(glucose-1, min(insulin+dose,9))
  else if (insulin > 0 and glucose > 0 and food == 0) then
    measure!(glucose-1).0.activity -> inject?dose ->
      PATIENT(glucose-1, insulin-1+dose)
  else
    measure!glucose.food.activity -> inject?dose ->
      PATIENT(glucose, min(insulin+dose,9))
```

Note that the use of ‘min’ and ‘max’ functions here is a necessary trick to prevent the Patient model from becoming infinite state, which would break the automated checks carried out by FDR. We impose sufficiently high, but abstract, upper bounds for levels of glucose and insulin, which we can show will not impact on the level of reality within this model.

The Monitor process retrieves information from the patient about glucose levels, food, and activity on every time interval. If the glucose level goes above a dangerous threshold where a ‘hyper’ could take place it will then execute an ‘alert’ event and then stop and deadlock the whole control system. (We should also check for ‘hypo’ events but that is left out here to preserve simplicity in this model.) The purpose of the alert would be to hand control over to the patient however the intention is that that should not happen as it would have already handed over the control to an algorithmic dosing system before the glucose level could ever rise that high.

An algorithmic system might not be as effective as an AI system in managing the stability of the insulin levels, because the AI system can potentially learn from its mistakes and converge towards an optimal dosing regime. But on the other hand, it will hopefully provide an upper bound as to the level of insulin that should be in the system within the constraints of how the simulated human body works.

```
MONITOR(overidden) =
  tock -> measure?glucose.food.activity ->
    if glucose > 8 then
      alert -> STOP
    else if (not overidden and glucose < 8) then
      signal_ai!glucose.food.activity -> MONITOR(false)
    else
      signal_alg!glucose.food.activity -> MONITOR(true)
```

We model the AI Pancreas algorithm as something that is completely unpredictable and potentially might perform any dangerous or malicious act. This is similar to how intruders are modelled in the CSP treatment of cryptographic protocol vulnerability by Lowe, Roscoe and others [10, 11, 12].

```
AI_PANCREAS =
  signal_ai?glucose.food.activity ->
    ((inject_ai!1 -> AI_PANCREAS) |~| (inject_ai!0 -> AI_PANCREAS))
```

On the other hand, we model the algorithmic Pancreas process as something which will always naively inject insulin if the residual blood sugar level is beyond a certain threshold and otherwise do nothing. This could result in a see-sawing effect in terms of the blood sugar in the system and one would hope that a truly intelligent AI system would learn to handle that better over time, however it does provide predictable safety behaviour.

```
ALG_PANCREAS =
  signal_alg?glucose.food.activity -> if (glucose > 3) then
    inject_alg!1 -> ALG_PANCREAS else inject_alg!0 -> ALG_PANCREAS
```

The Injector process may receive instructions from either the AI process or the algorithmic process to inject insulin into the patient, which it will always follow.

```
INJECTOR =
  (inject_ai?x -> inject!x -> INJECTOR) []
  (inject_alg?x -> inject!x -> INJECTOR)
```

Now we compose all these components in parallel to create the CSP process network. (See figure 2 for the process diagram.)

```
SYSTEM =
  ((AI_PANCREAS ||| ALG_PANCREAS) [|{|inject_ai, inject_alg|}|] INJECTOR)
      [|{|signal_alg, signal_ai, inject|}|]
  (MONITOR(false) [|{|measure, tock|}|] PATIENT(5,1))
```

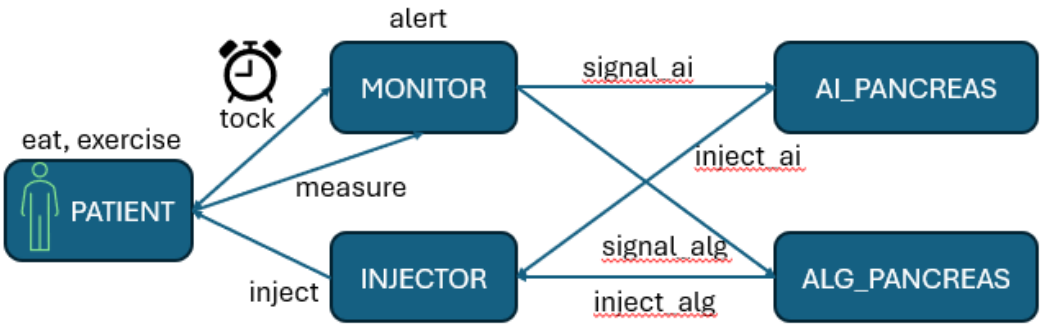

Figure 2. Communication Diagram.

We assert that this system is deadlock free in the failures-divergences model which 'means that the monitor never will trigger the 'alert' event and suspend the whole system – the safety system (algorithmic process) is doing its job.

```
assert SYSTEM :[deadlock free [FD]]
```

For the sake of completeness, we should also test this system with the safety system disabled. We change the monitor so that it always relies upon the potentially malicious AI algorithm.

```
UNSAFE_MONITOR =
  tock -> measure?glucose.food.activity ->
    if glucose > 8 then
       alert -> STOP
    else
      signal_ai!glucose.food.activity -> UNSAFE_MONITOR

UNSAFE_SYSTEM =
  ((AI_PANCREAS ||| ALG_PANCREAS) [|{|inject_ai, inject_alg|}|] INJECTOR)
    [|{|signal_alg, signal_ai, inject|}|]
  (UNSAFE_MONITOR [|{|measure, tock|}|] PATIENT(5,1))
```

We then run the same assertion on the unsafe system and we would expect to see this fail.

```
assert UNSAFE_SYSTEM :[deadlock free [FD]]
```

Figure 3 shows what we see within the FDR tool console when we load the CSP code and run the assertions. The system reports as we would expect that the first system is deadlock free and therefore it prevents the adverse hyperglycaemic event from happening.

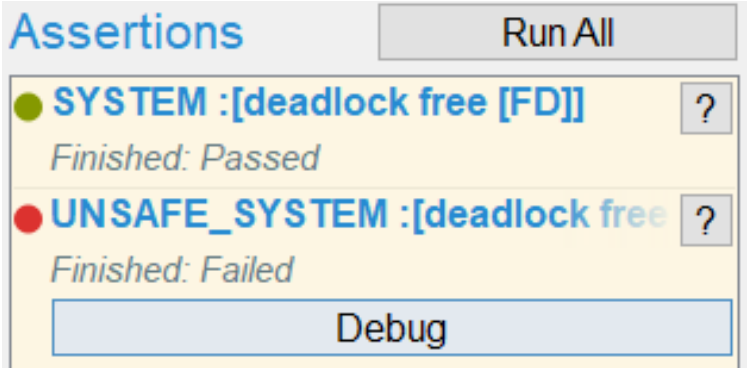

Figure 3. FDR Console

However the unsafe system which relies purely on the AI process, might deadlock and this corresponds to a serious adverse event occuring. A minimal sequence of events which triggers this is shown within the tool. It corresponds to the patient repeatedly eating and the AI system withholding any top-up insulin injections so that the blood sugar continues to rise until it hits a dangerous level

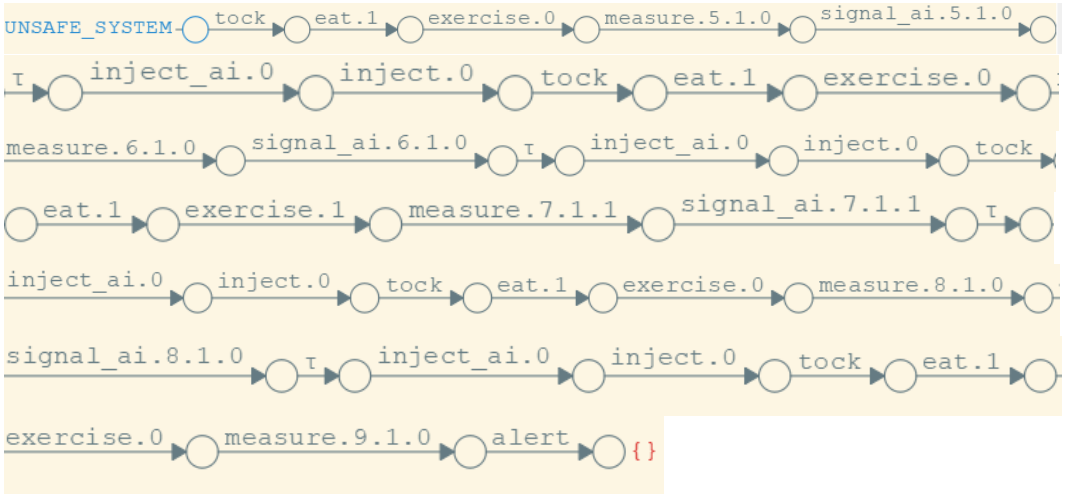

Figure 4. FDR reports deadlock event trace.

4 Conclusions

Despite the simplicity of the model presented, I believe it provides encouragement that we could facilitate the design of guardrails for AI systems, through this style of analysis. It should be worthy of future research. Clearly its accuracy and success are very much dependent on how realistically we can model the environment. In the case study illustrated this was the process representing how a patient's blood sugar and insulin levels respond over time to the three stimuli of eating, exercising and receiving

insulin injections. These were massively simplified in this example and to create a more functional model would require substantial domain knowledge of Human Physiology and Biology, especially in the context of type 1 diabetes.

Please note that the opinions expressed here are those of the author and do not necessarily represent those of Lloyds of London.

5 References

1. "Policy paper. Introducing the AI Safety Institute", https://www.gov.uk/government/publications/ai-safety-institute-overview/introducing-the-ai-safety-institute
2. Jeremy M. R. Martin. "Concurrency and models of abstraction: past, present and future", Proceedings of 2023 IEEE Concurrent Processes Architectures and Embedded Systems Virtual Conference (COPA 2023).
3. Liu C, Avari P, Leal Y, Wos M, Sivasithamparam K, Georgiou P, Reddy M, Fernández-Real JM, Martin C, Fernández-Balsells M, Oliver N, Herrero P. "A Modular Safety System for an Insulin Dose Recommender: A Feasibility Study", J Diabetes Sci Technol. 2020 Jan;14(1):87-96. doi: 10.1177/1932296819851135. Epub 2019 May 22. PMID: 31117804; PMCID: PMC7189144.
4. C. A. R. Hoare. "Communicating sequential processes", Prentice-Hall, 1985.
5. A. W. Roscoe. "The theory and practice of concurrency", Prentice Hall, 1998.
6. A. W. Roscoe. "Understanding concurrent systems", Springer, 2010.
7. Brookes, Stephen D., and A. W. Roscoe. "CSP: A practical process algebra, Theories of Programming: The Life and Works of Tony Hoare", 2021. 187–222.
8. Thomas Gibson-Robinson, Guy Broadfoot, Gustavo Carvalho, Philippa Hopcroft, Gavin Lowe, Sidney Nogueira, Colin O'Halloran, and Augusto Sampaio. "FDR: from theory to industrial application", In Concurrency, Security, and Puzzles, pages 65–87. Springer, 2017.
9. M. Goldsmith and J. Martin. "Parallelization of FDR", in Workshop on Parallel and Distributed Model Checking, affiliated to CONCUR 2002 (13th International Conference on Concurrency Theory), Brno, Czech Republic, 2002.
10. Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR, in International Workshop on Tools and Algorithms for the Construction and Analysis of Systems, pages 147–166. Springer, 1996. 29.
11. Gavin Lowe. Casper: a compiler for the analysis of security protocols, Journal of computer security, 6(1-2):53–84, 1998.
12. Peter Ryan, Steve A Schneider, Michael Goldsmith, Gavin Lowe, and A.W. Roscoe. The modelling and analysis of security protocols: the CSP approach, Addison-Wesley Professional, 2001.