APPENDIX
PROMPTS USED FOR THE LEVEL-CROSSING EXAMPLE

The following prompt was used to generate the code presented in Listing 1. We asked the LLM to generate a non-deterministic function since the requirements also define a non-deterministic system. For example, another train can approach before or after the barriers are lowered.

```
The following is a system of a controller for a gate at a
     railway crossing - an intersection between a railway
     line and a road at the same level.

Events: Approaching, Entering, Leaving, Lower, Raise

Requirements:

1. When a train passes, the sensor system activates the
     exact event order: approaching, entering, and leaving.
2. The barriers are lowered when a train approaches and
     then raised.
3. A train may not enter while barriers are raised.
4. The barriers may not be raised while a train is passing,
      i.e. it approached but did not leave.

Based on the following desired events and requirements,
     build a function that satisfies them.
The function should return a list of events that satisfies
     the requirements.
The function should be non-deterministic and should be able
      to generate a different list of events each time it is
      called.
Additionally, the function should terminate.



Please answer with the code only and without any additional
      text.
```

The prompt used to generate the code presented in Listing 5:

```
Behavioral Programming (BP) is a modeling paradigm designed
      to allow developers to specify the behavior of
     reactive systems incrementally and intuitively in a way
      that is aligned with system requirements.
A BP program, is defined by a set of b-threads representing
      the different behaviors of the system.
The protocol involves each b-thread issuing a statement
     before selecting every system-generated event.
In the statement, the b-thread declares which events it
     requests, waits for (but does not request), and blocks
     (forbids from happening).
After submitting the statement, the b-thread is paused.
When all b-threads have submitted their statements, we say
     the b-program has reached a synchronization point (
     yield point).
Then, an event arbiter picks a single event that has been
     requested but not blocked.
The b-program then resumes all b-threads that either
     requested or waited for the chosen event, leaving
     others paused, and their existing statements are
     carried forward to the next yield point.
This process is repeated throughout the program’s execution
     , terminating when there are no requested non-blocked
     events.

For instance, given the consider the following system
     requirements that controls hot and cold water taps,
     whose output flows are mixed:

Events: HOT, COLD

Requirements:
1. Do ’HOT’ three times.
2. Do ’COLD’ three times.
3. Prevent ’HOT’ from being executed consecutively.

The following is a b-program that satisfies the
     requirements:
@b_thread
def req_1():
    for i in range(3):
        yield {request: BEvent("HOT")}

@b_thread
def req_2():
    for i in range(3):
        yield {request: BEvent("COLD")}

@b_thread
def req_3():
    while True:
        yield {waitFor: BEvent("HOT")}
        yield {block: BEvent("HOT"), waitFor: BEvent("COLD
     ")}

Based on the following desired events and requirements,
     build a b-program that satisfies them.

The following is a system of a controller for a gate at a
     railway crossing - an intersection between a railway
     line and a road at the same level.

Events: Approaching, Entering, Leaving, Lower, Raise

Requirements:

1. When a train passes, the sensor system activates the
     exact event order: approaching, entering, and leaving.
2. The barriers are lowered when a train is approaching and
      then raised.
3. A train may not enter while barriers are raised.
4. The barriers may not be raised while a train is passing,
      i.e. it approached but did not leave.

Create a separate b-thread for each requirement. Avoid
     including any extra keys in yield statements beyond
     request, waitFor, and block. Please answer with the
     code only and without any additional text.
```