



REYKJAVÍK UNIVERSITY

T-419-CADP - ASSIGNMENT 1

Three Dogs and a Garden

Freyr Elí Sveinbjörnsson
freyr23@ru.is

Georg Ingi Einarsson
georg23@ru.is

Óliver Máni Samúelsson
olivers23@ru.is

Instructor: Marcel Kyas

January 30, 2026

1 Solution Description

In order to fix the problem of multiple conflicts we must introduce gates. Boolean flags are sufficient for two processes because only one conflict exists, but with three processes two conflicts must be resolved, requiring multiple gates. The gates work as a kind of filter, always letting at most $N - 1$ dogs through, leaving one dog behind at each gate. In this case we would need two gates because there are 3 dogs. In order to decide which dogs get to pass through we need to have private variables for each dog.

Let us say all three dogs try to enter at the same time. In order to decide who gets to pass, each dog writes to its own private variable at each gate level. The dog that writes last to its private variable must wait at the gate while the others pass. The dogs cannot write at the exact same time because it depends on the order in which the processes are scheduled to execute the write.

When a dog reaches a gate, it sets its private `level` variable to indicate which gate it has reached. For example, `level = 1` means the dog is waiting at the first gate, and `level = 2` means the dog is waiting at the second gate, just before entering the garden. Each dog also writes its own ID to its private `last[gate]` variable when it reaches that gate.

After a dog leaves the garden, it resets its `level` variable to 0, indicating that it is no longer competing. Dogs waiting at a gate continuously observe the `level` variables of other dogs, and once the blocking condition is removed (no other dogs at higher levels and this dog was not last to write), they may proceed through the gate.

Each dog has private variables that only it can write to, though other dogs may read them. Specifically, each dog i has:

- `level[i]`: The current gate level dog i is at (0 means not competing)
- `last[i][gate]`: Dog i 's ID written when reaching each gate

The key insight is that by having each dog write to its own private variable and read others' variables, we can determine which dog wrote last without needing a shared variable that multiple processes write to. This satisfies the requirement of dynamic private access; only one process writes to each variable at runtime.

2 Pseudocode

Three Dogs Problem - Pseudocode

Private Variables (for each dog i)

- `level[i] = 0` : Current gate level
- `last[i][1] = 0` : Dog i's ID when reaching gate 1
- `last[i][2] = 0` : Dog i's ID when reaching gate 2

Constants

- `N = 3` : Number of dogs
- `GATES = 2` : Number of gates (N-1)

Process for Dog i

```
process Dog_i:
  while true:
    // Entry Protocol
    for gate = 1 to GATES:
      level[i] = gate
      last[i][gate] = i

      while (exists j ≠ i: level[j] >= gate) AND
        (forall j ≠ i: last[j][gate] == j AND level[j] >= gate):
        wait

      // Critical Section
      play_in_garden()

      // Exit Protocol
      level[i] = 0
```

Example: Alice's Process

```
process Alice:
  while true:
    // Gate 1
    level[Alice] = 1
    last[Alice][1] = Alice
    while (level[Bob] >= 1 OR level[Charlie] >= 1) AND ...
      wait

    // Gate 2
    level[Alice] = 2
    last[Alice][2] = Alice
    while (level[Bob] >= 2 OR level[Charlie] >= 2) AND ...
      wait

    play_in_garden()
    level[Alice] = 0
```

Figure 1: Pseudocode for three dogs and a garden

3 Correctness Arguments

3.1 Mutual Exclusion: At Most One Dog in the Garden

A dog can enter the garden only after passing the final gate (gate 2). At this gate, each dog writes to its own private $\text{last}[i][2]$ variable. Assume, for contradiction, that two dogs are inside the garden simultaneously. Then both must have passed gate 2.

Consider the dog that wrote to its last variable most recently at gate 2. This dog would have seen the other dog's level at 2 or higher **and** seen that the other dog had already written its ID to its $\text{last}[2]$ variable. Therefore, the waiting condition would have been satisfied, forcing this dog to wait. This contradicts our assumption that it entered the garden.

Hence, at most one dog can be in the garden at any time.

3.2 Deadlock Freedom

If we suppose some dogs want to enter, we look at the highest level value reached by any dog. At that level, all dogs present are competing. Among these dogs, exactly one will be the last to write to its $\text{last}[\text{level}]$ variable.

The dog that wrote last will wait, but all other dogs at that same level will see that they were not last (they can observe the last dog's write to its last variable). Therefore, at least one dog always makes progress past this level.

Repeating this argument, one dog must eventually pass the final level and enter the garden. So it is impossible for all dogs to be permanently stuck.

3.3 Starvation Freedom

A dog can only be blocked at a gate if it was the last to write to its $\text{last}[\text{gate}]$ variable while other dogs are at that gate or beyond. Dogs ahead (with level values at the current gate or higher) will eventually enter the garden, leave, and reset their level to 0.

Once a dog resets its level to 0, it no longer blocks anyone at any gate because the waiting condition checks $\text{level}[j] \geq \text{gate}$. This means a waiting dog will eventually see no other dogs with $\text{level} \geq \text{gate}$, allowing it to proceed.

This happens at each gate, so the dog eventually passes all gates and enters the garden. Therefore, no dog can starve indefinitely.