Challenge 3: Static Tree Barriers

This challenge focuses on multi-threaded executions and targets verifiers for concurrent programs. However, the problem can also be encoded as a transition system and verified by a verifier for sequential code.

Consider a multi-threaded program execution. In each state with N threads, we assume we have given a binary tree with N nodes. Each node corresponds to exactly one thread. In the context of this challenge, we do not consider thread creation and termination; the number of nodes and their position in the tree is assumed to be immutable. Each node has references to at most two children, and each node except for the root has a reference to its parent. Moreover, each node stores a boolean value sense and an integer version. Hence, we have the following data structure:

```
class Node {
    final Node left, right;
    final Node parent;

    boolean sense;
    int version;

    // methods and constructors omitted
}
```

The tree described above represents a tree barrier that can be used for thread synchronization. This synchronization is performed by the following method of class Node:

```
void barrier()
    requires !sense
    ensures  !sense
{
    // synchronization phase
    if(left != null)
    while(!left.sense) { }

    if(right != null)
    while(!right.sense) { }

    sense = true;  // assume this statement and the next to execute
    version++;     // simultaneously (that is, in one step)

    // wake-up phase
    if(parent == null)
```

```
        sense = false;

        while(sense) { }

        if(left != null)
        left.sense = false

        if(right != null)
        right.sense = false
}
```

Synchronization is performed in two phases. In the synchronization phase, each thread that calls `barrier()` on its node waits until all threads have called barrier. The `sense` field is used to propagate information about waiting threads up in the tree. When all threads have called `barrier()`, the propagation reaches the root of the tree and initiates the wake-up phase, which proceeds top-down.

Assume a state in which `sense` is false and `version` is zero in all nodes. Assume further that no thread is currently executing `barrier()` and that threads invoke `barrier()` only on their nodes. The number of threads (and, thus, the number of nodes in the tree) is constant, but unknown.

Tasks. Prove that:

1. the following invariant holds in all states:
   If `n.sense` is true for any node `n` then `m.sense` is true for all nodes `m` in the subtree rooted in `n`.
2. for any call `n.barrier()`, if the call terminates then there was a state during the execution of the method where all nodes had the same version.

Hint: The problem can be solved with a verifier for sequential programs by encoding it as a non-deterministic transition system that represents threads and their states explicitly in each state of the transition system.