Design Document – Scott Parkinson

The tree generation program as of the 18th of August performs the following sequence of events.

1. Variable house keeping
2. Generate all world states through binary counting
3. Generate Actions and Action sequences
4. Sort Sequences
5. Build Tree

In many cases these steps have been designed to be interchangeable in order of execution with varied methodologies. For example we have different ways of generating trees, generating actions (and in the future the sequences, specifically their length) and the sorting of the sequences. Understanding that requirement could change lead to this design approach.

The program is executed by evoking the main execution in a GoalPlanBuilder object. An instantiation of a GoalPlanTree will contain several variables, listed below:

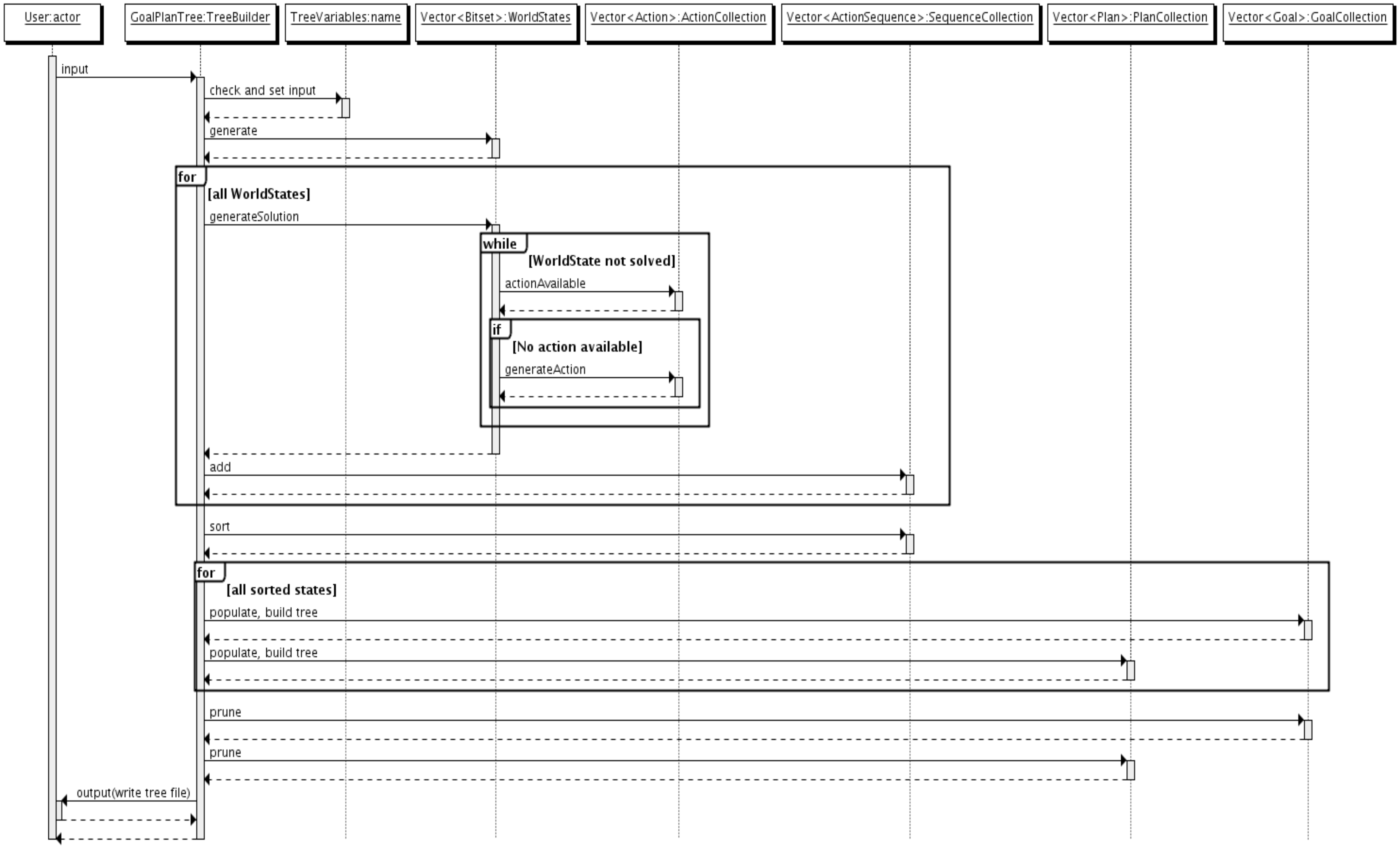
- Depth: Integer
- Number of Attributes: Integer
- Breadth: Integer
- Number of Preconditions: Integer
- Goal Mask: String

- Attribute Collection: Vector<Attribute>
- Action Collection: Vector<Action>
- Sequence Collection: Vector<ActionSequence>
- Sorted Sets: Vector <Vector<Vector<Action>>>
- Plan Collection: Vector<Plan>
- Goal Collection: Vector<Goal>
- Bitset Collection : Vector

- Goal State: BitSet
- Top Level Goal: Goal
- Output File: String

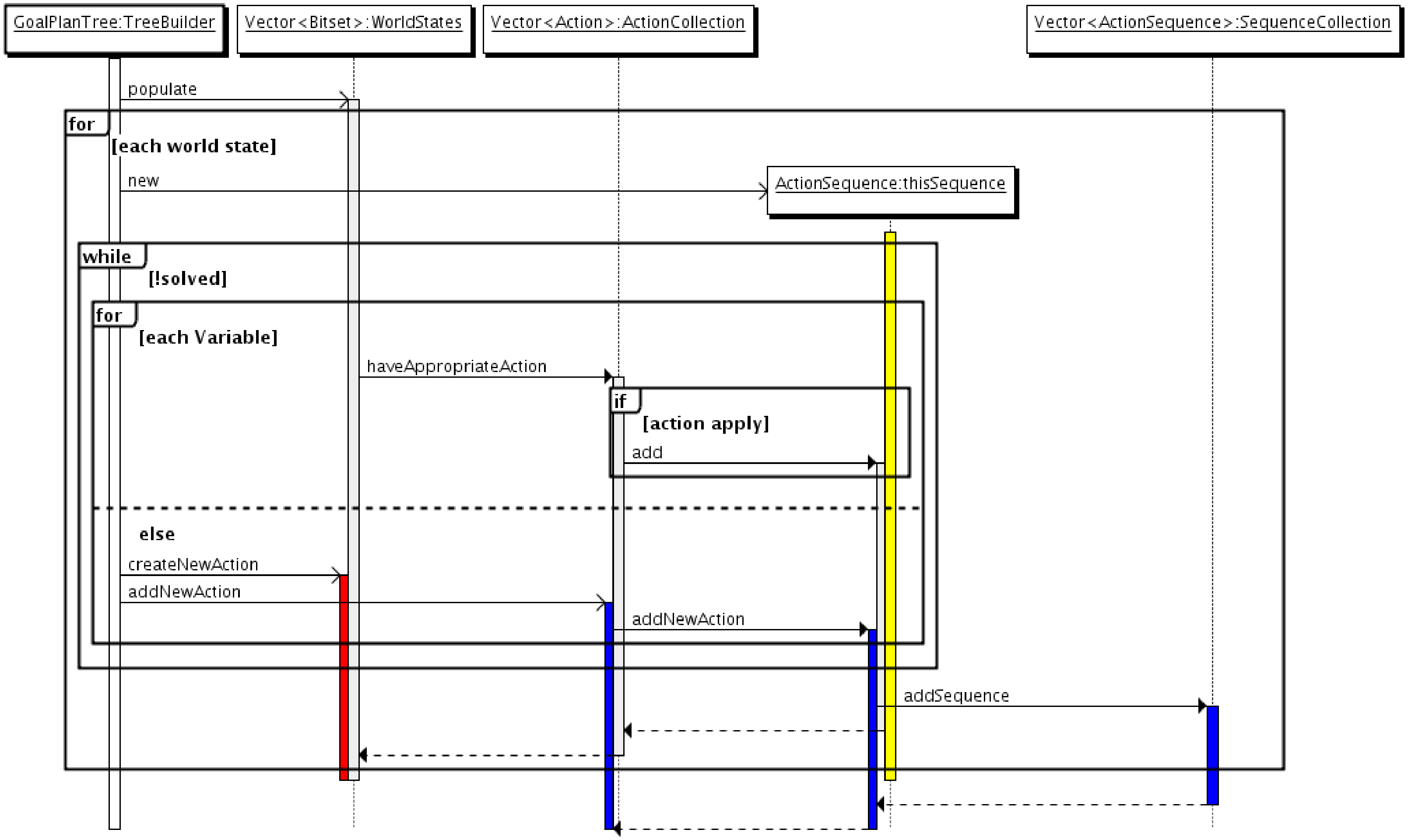
These will be set at various states depending on both user input and the order of execution of methods.

Operations Procedure - Overview



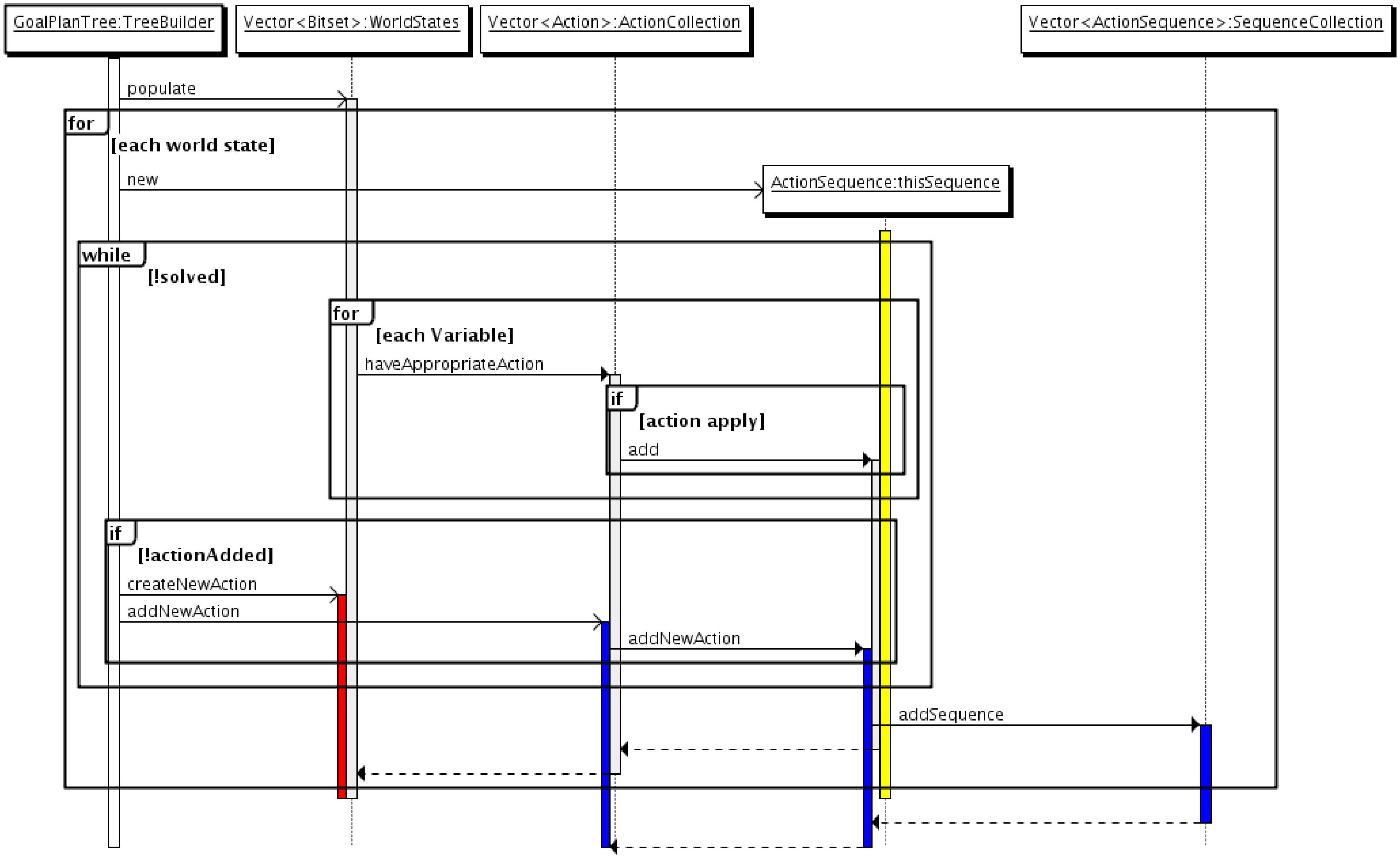
The above digram give an overview of the order of execution for the Tree Builder.

Action Generation and Sequence Construction – Lazy



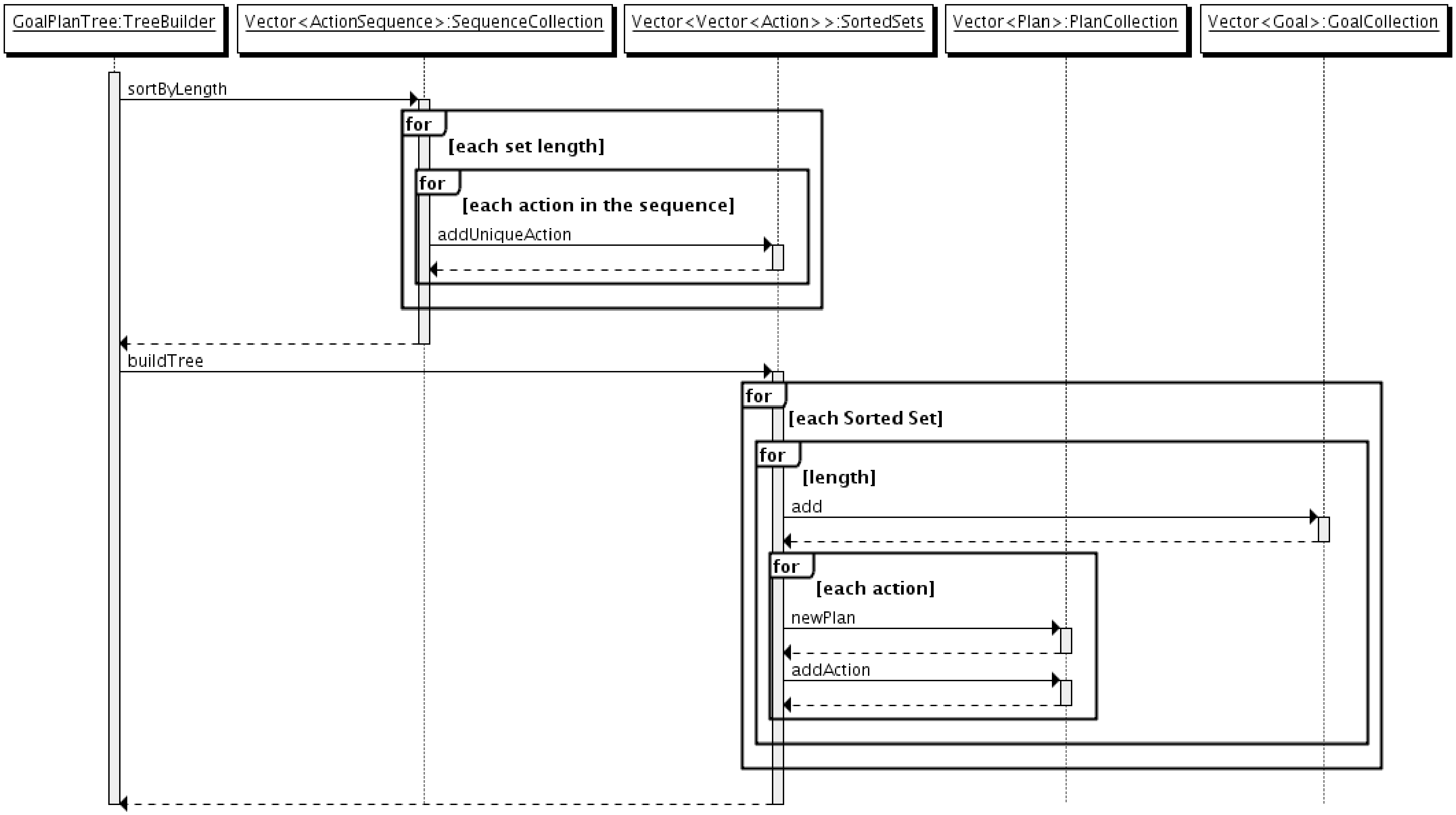
The above diagram shows one of the processes used to generate both actions and sequences. It has been dubbed lazy as it will generate an action as soon as it is unable to change a bit.

Action Generation and Sequence Construction – Smart



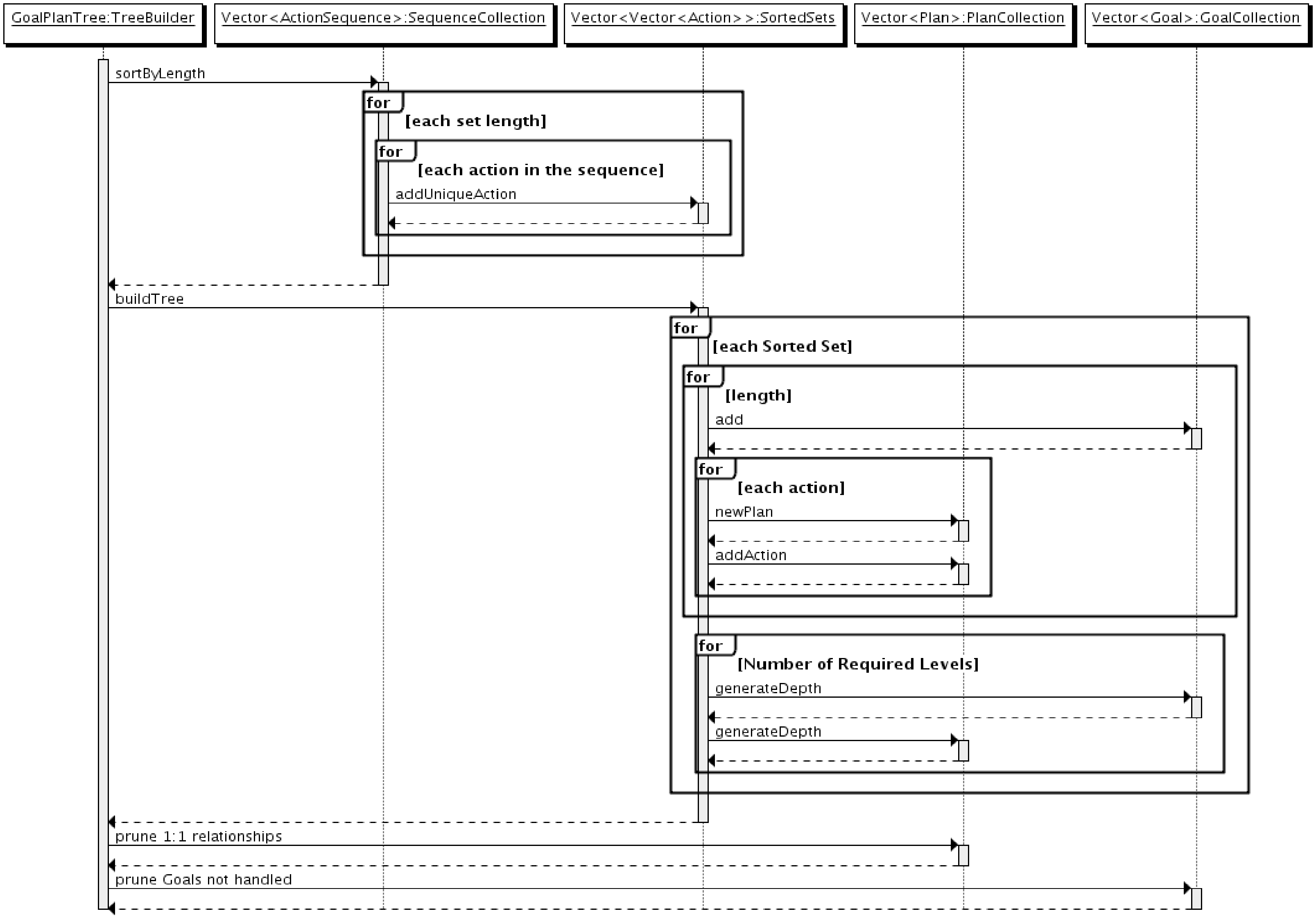
The above diagram shows the second process that can be used to generate both actions and sequences. This process differs from the lazy approach by trying to change an bits that require change before generating a new action.

Flat-OR Tree Generation


GoalPlanTree:TreeBuilder
Vector<ActionSequence>:SequenceCollection
Vector<Vector<Action>>:SortedSets
Vector<Plan>:PlanCollection
Vector<Goal>:GoalCollection
sortByLength
for
[each set length]
for
[each action in the sequence]
addUniqueAction
buildTree
for
[each Sorted Set]
for
[length]
add
for
[each action]
newPlan
addAction

The above Diagram shows the process used to generate goal plan trees with a depth of only two.

Drop-OR – Tree Generation



This method works by generating the requested depth and attaching the plans with actions below. This method needs to be refined as the tree structure breaks the goals.

Data Design

This section outlines the data structures used in the Goal Plan Tree Builder.

Action: This object is used to define an action in our tree generation program. The following variables are part of the structure
Contents
Pre Statement: String
Post Statement: String
Name : String
Transformer: Change Description
String Representation
Number of Attributes

Action Sequence: Used to store sets of actions in order of execution.
Sequence : Vector
Start State: Bitset
Number of Attributes

Attribute: Defines an attribute in the tree. Developed for use as either boolean of multiple variable attribute.
Observable : Boolean
Boolean attribute : Boolean
Name : String
Length : Integer
Set Values : String Array

Goal: Defines a Goal in our tree. Implements the TreeGenElement interface. This object is fairly light, but contains methods to
ID: String
depth: The depth of this tree in

Plan: The definition of a Plan for use in the program. Implements the TreeGenElement interface.
ID: String
Depth: Integer
Handle Goal: String
pre state: boolean
init pre state: boolean
bodySpace: Vector<TreeGenElement>

TreeGenElement: Interface for Goals and Plans.