# Domain Descriptions

October 5, 2023

# 1 Appendix

## 1.1 Car Domain

The planning problem consists of a car (initially at rest) that has to cover a given distance within a specified time bound. Further, it is needed to be ensured that the car has zero velocity at the end as well. The domain comprises of six function symbols: $d$, $v$, $a$, *upLimit*, *downLimit* and $runningTime$, three predicate symbols: *running*, *engineBlown* and *goalReached*, three instantaneous actions *accelerate*, *decelerate* and *stop*, one event *engineExplode* and one process *moving*. The function symbols $d$, $v$, and $a$ respectively represent the distance, velocity, and acceleration of the car, whereas *upLimit* and *downLimit* specify the upper and lower bounds on the acceleration. $runningTime$ encodes the elapsed time of the car. The predicates *running*, *engineBlown*, and *goalReached* define the states of the car. The actions *accelerate* and *decelerate* respectively increase and decrease the acceleration of the car by one unit. The *stop* action is applicable when the car covers the specified distance and has a velocity of zero. The event *engineExplode* is triggered when the velocity and acceleration of the car reach a certain threshold and the car is in the *engineBlown* state. The process *moving* is activated when the car is in the *running* state specified by a predicate *running*. This process updates the distance and the velocity of the car as a function of time. Initially, the car is in the *running* state with zero velocity. The goal is to traverse a distance of 30 units within 50 units of time. The planning domain represented in PDDL+ is shown below.

```
(define (domain car)
(:requirements :typing :durative-actions :fluents :time
  :negative-preconditions :timed-initial-literals)
(:predicates (running) (stopped) (engineBlown)
  (goal_reached))
(:functions (d) (v) (a) (up_limit) (down_limit)
  (running_time))
(:process moving
:parameters ()
:precondition (and (running))
:effect (and (increase (v) (* #t (a))) (increase (d)
  (* #t (v))) (increase (running_time) (* #t 1))))
(:action accelerate
  :parameters()
  :precondition (and (running) (< (a) (up_limit)))
  :effect (and (increase (a) 1)))
(:action decelerate
  :parameters()
  :precondition (and (running) (> (a) (down_limit)))
  :effect (and (decrease (a) 1)))
(:event engineExplode
:parameters ()
:precondition (and (running) (>= (a) 1) (>= (v) 100))
:effect (and(not(running))(engineBlown)(assign (a)0)))
(:action stop
:parameters()
:precondition(and(=(v) 0)(>=(d) 30)(not(engineBlown)))
:effect (goal_reached)))
```

One of the problem instances we use is shown below where the initial condition *init* specifies that the predicate *running* is initially *true*, and the initial values of the functions $a$, $v$, $d$, and $runningTime$ are assigned 0 while *up_limit* and *down_limit* are set to 2 and -1 respectively. The goal for the car is to complete the journey within 50 units of time, without exploding the engine.

```
(define (problem car_prob)
(:domain car)
(:init (running) (= (running_time) 0) (= (up_limit) 2)
  (= (down_limit) -1) (= d 0) (= a 0) (= v 0))
(:goal (and(goal_reached) (not(engineBlown))
  (<= (running_time) 50)))
(:metric minimize(total-time)))
```

One instance of the plan for this planning problem generated by SMT-Plan+ is as follows:

[basicstyle=, label=Plan] (@**Time**@) (@**Action**@) (@**Duration**@) 0.0: (accelerate) [0.0] 1.0: (decelerate) [0.0] 31.0: (decelerate) [0.0] 32.0: (stop) [0.0] (@***makespan:***= 32.0 units.@) Each line contains three tuples: the time when the planner triggered the action, the action itself, and the duration for which the action is active. Duration tuple 0 signifies the action is instantaneous in nature. The *makespan* of the plan is 32.0 units of time and the plan-length is 4.

## 1.2 Generator-events Domain

The generator-events domain consists of a generator that has to run for a specified amount of time. The domain comprises five function symbols: *fuelLevel*, *capacity*, *fuelInTank*, *ptime* and *dur*, five predicate symbols: *generator-ran*, *available*, *using*, *safe* and *generatorStarted*, three instantaneous actions *refuel*, *generateStart* and *generateEnd*, two processes *refueling* and *generateProcess*, and three events: *tankEmpty*, *generatorOverflow* and *generateFail*. We need to instantiate objects of types generator and tank, which will be set as parameters to different domain symbols while respecting their arities. The *fuelLevel* and the *capacity* are associated with the generator and indicate the current fuel level and fuel storage capacity of the generator, whereas the *fuelInTank* and the *ptime* are associated with the tanks, indicating fuel reserve in a tank and fuel pouring time from a tank while in use, respectively. The *dur* measures the duration of time the generator has run. The *generator-ran* indicates whether the generator ran successfully for a specified amount of time (1000 time units), *available* indicates tanks that are in store for use, *using* indicates tanks that are being used by the generator, *safe* indicates if the generator is operating safely, and *generatorStarted* indicates that the generator has started it's operation. The instantaneous-action *generateStart* has preconditions *fuelLevel* $\geq 0$, *safe* and *not(generatorStarted)* to indicate that the current fuel level in the generator needs to be greater than 0, the generator needs to be in the safe mode, and the generator not already activated. It has the effect to make *generatorStarted* true to indicate that the generator is activated and initializes *dur* to 0. *generateStart* triggers the process *generateProcess* decreases the *fuelLevel* in the generator at a constant rate and increases the value of *dur* at a rate of 1. The value of *dur* signifies the generator activation time. The instantaneous-action *generateEnd* has preconditions *generatorStarted*, *fuelLevel* $\geq 0$, *safe* and *dur* $= 1000$ to indicate that the generator is activated, the current fuel level in the generator needs to be greater than 0, the generator is in the safe mode and has run for a duration of 1000 time units. It has the effect to make *generator-ran* to true and *generatorStarted* to false to indicate that the generator ran successfully for the duration of 1000 time units and now is deactivated. *generateEnd* stops the *generateProcess*. The instantaneous-action *refuel* enables the generator to use an available tank and activates the process *refueling* which in turn increases the *fuelLevel* at a rate proportional to the pouring time *ptime* while decreasing the *fuelInTank* at the same rate. The process *generateProcess* decreases the *fuelLevel* in the generator at a constant rate and increases the time *dur* which The event *emptyTank* triggers to indicate that a tank is unavailable for use when *fuelInTank* becomes 0 for it, whereas the event *generatorOverflow* drives the generator to an unsafe-mode when its *fuelLevel* exceeds the *capacity* threshold. The event *generateFail* occurs when the generator is unable to run for the specified duration of time of 1000 units. The planning domain represented in PDDL+ is shown below.

```
(define (domain generatorplus)
(:requirements :fluents :adl :typing :time
  :negative-preconditions)
(:types generator tank)
(:predicates (generator-ran) (available ?t - tank)
  (using ?t - tank ?g - generator) (safe ?g - generator)
  (generatorStarted ?g - generator))
(:functions (fuelLevel ?g - generator)
  (capacity ?g - generator) (fuelInTank ?t - tank)
  (ptime ?t - tank) (dur ?g - generator))
(:action generateStart
 :parameters (?g - generator)
 :precondition (and (not (generatorStarted ?g))
  (>= (fuelLevel ?g) 0) (safe ?g))
 :effect (and (generatorStarted ?g)
  (assign (dur ?g) 0)))
(:process generateProcess
 :parameters (?g - generator)
 :precondition (and (generatorStarted ?g))
 :effect (and (decrease (fuelLevel ?g) (* #t 1))
    (increase (dur ?g) (* #t 1))))
(:event generateFail
 :parameters (?g - generator)
 :precondition (and (generatorStarted ?g)
   (not (= (dur ?g) 1000)) (not (>= (fuelLevel ?g) 0)))
 :effect (and (assign (dur ?g) 1001)))
(:action generateEnd
 :parameters (?g - generator)
 :precondition (and (generator started ?g)
   (>= (fuelLevel ?g) 0) (safe ?g) (= (dur ?g) 1000))
 :effect (and (generator-ran)
   (not (generatorStarted ?g))))
(:action refuel
 :parameters (?g - generator ?t - tank)
 :precondition (and (not (using ?t ?g)) (available ?t))
 :effect (and (using ?t ?g) (not (available ?t))))
(:process refuelling
 :parameters (?g - generator ?t -tank)
 :precondition (and (using ?t ?g))
 :effect (and (decrease (fuelInTank ?t) (* #t
   (* 0.001 (* (ptime ?t) (ptime ?t))))) (increase
   (ptime ?t) (* #t 1)) (increase (fuelLevel ?g)
   (* #t (* 0.001 (* (ptime ?t) (ptime ?t)))))))
(:event tankEmpty
 :parameters (?g - generator ?t - tank)
 :precondition (and (using ?t ?g)
   (<= (fuelInTank ?t) 0))
 :effect (and (not (using ?t ?g)) ))
(:event generatorOverflow
 :parameters (?g - generator)
 :precondition (and (> (fuelLevel ?g) (capacity ?g))
   (safe ?g))
 :effect (and (not (safe ?g)))))
```

For problem instance 1, the initial state specifies that the current *fuelLevel* of the generator *gen* is 940 units, whereas the *capacity* to hold fuel in the generator is 1600 units. The tanks *available* to use are *t1* and *t2*, and the fuel reserve in each tank is 40 units. Initially, the generator is in the *safe* mode. The predicate and function symbols that are not initialized explicitly in the initial state are initialized with a default value of false and 0, respectively. The goal condition comprises a single constraint *(generator-ran)* which needs to be true in the goal state indicating that the generator ran successfully for a specified amount of time (1000 time units).

```
(define (problem run-generatorplus)
 (:domain generatorplus)
 (:objects gen - generator tank1 tank2 - tank)
 (:init (= (fuelLevel gen) 940) (= (capacity gen) 1600)
   (= (fuelInTank tank1) 40) (= (fuelInTank tank2) 40)
   (available tank1) (available tank2) (safe gen))
 (:goal (generator-ran)))
```