PDDL

Exercises

Matteo Cardellini

Università degli Studi di Genova

Numeric Expressions

Write the following expressions as PDDL numeric expressions using the LISP-like sintax:

$$x := x + y + z \tag{1}$$

$$y := (x + y * z) - (y - z) * 3 \tag{2}$$

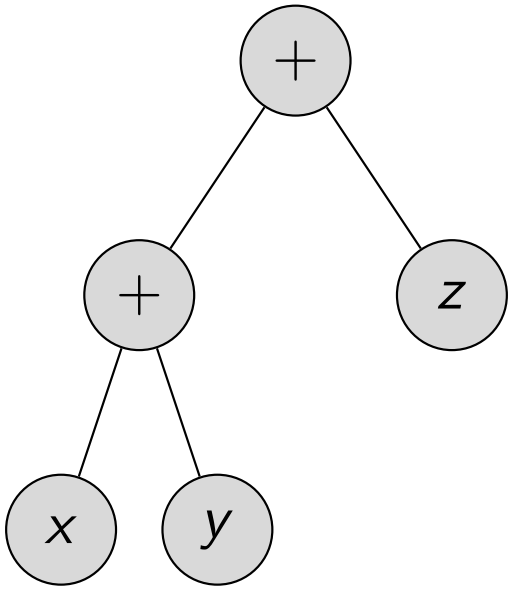
$$y := z^3 + 4z^2 + 3z + 2 \tag{3}$$

$$\frac{dx}{dt} = \frac{(100 - x) \cdot y}{z} \tag{4}$$

Numeric Expressions

$$x := x + y + z$$

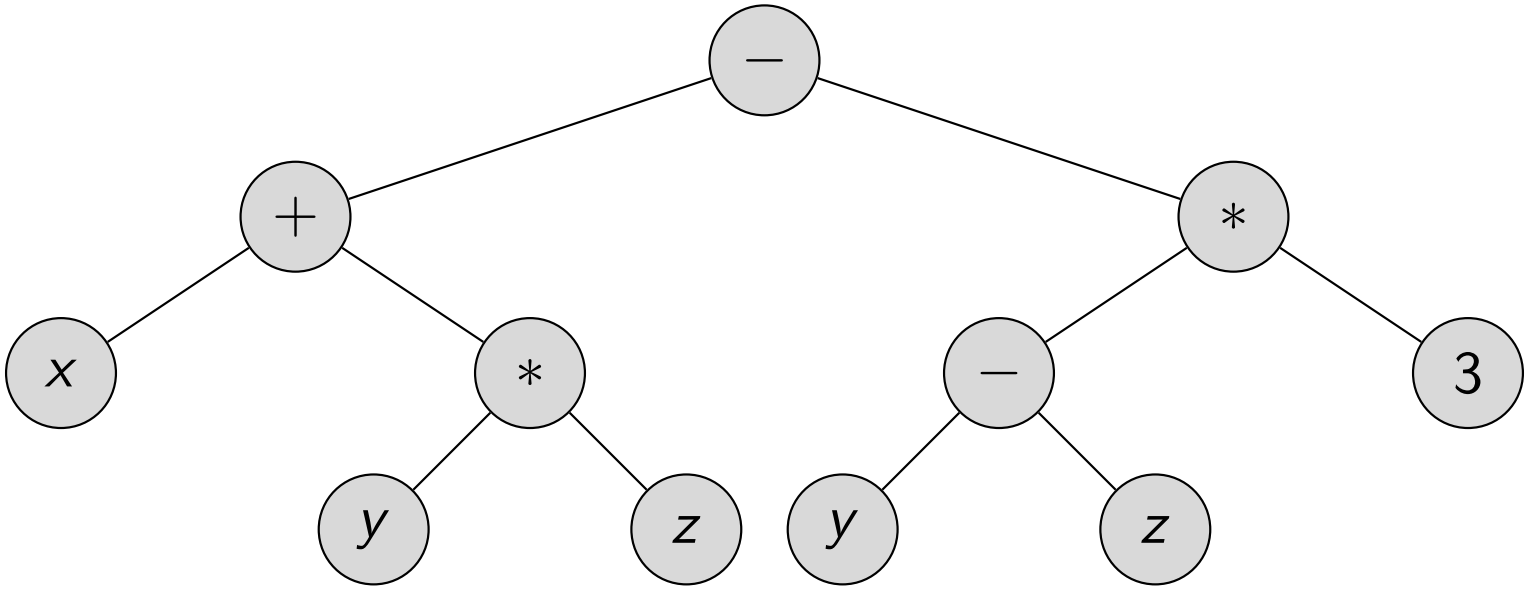
$$x := (x + y) + z$$




```
(increase (x) (+ (y) (z)))
```

Numeric Expressions

$$y := (x + y * z) - (y - z) * 3$$

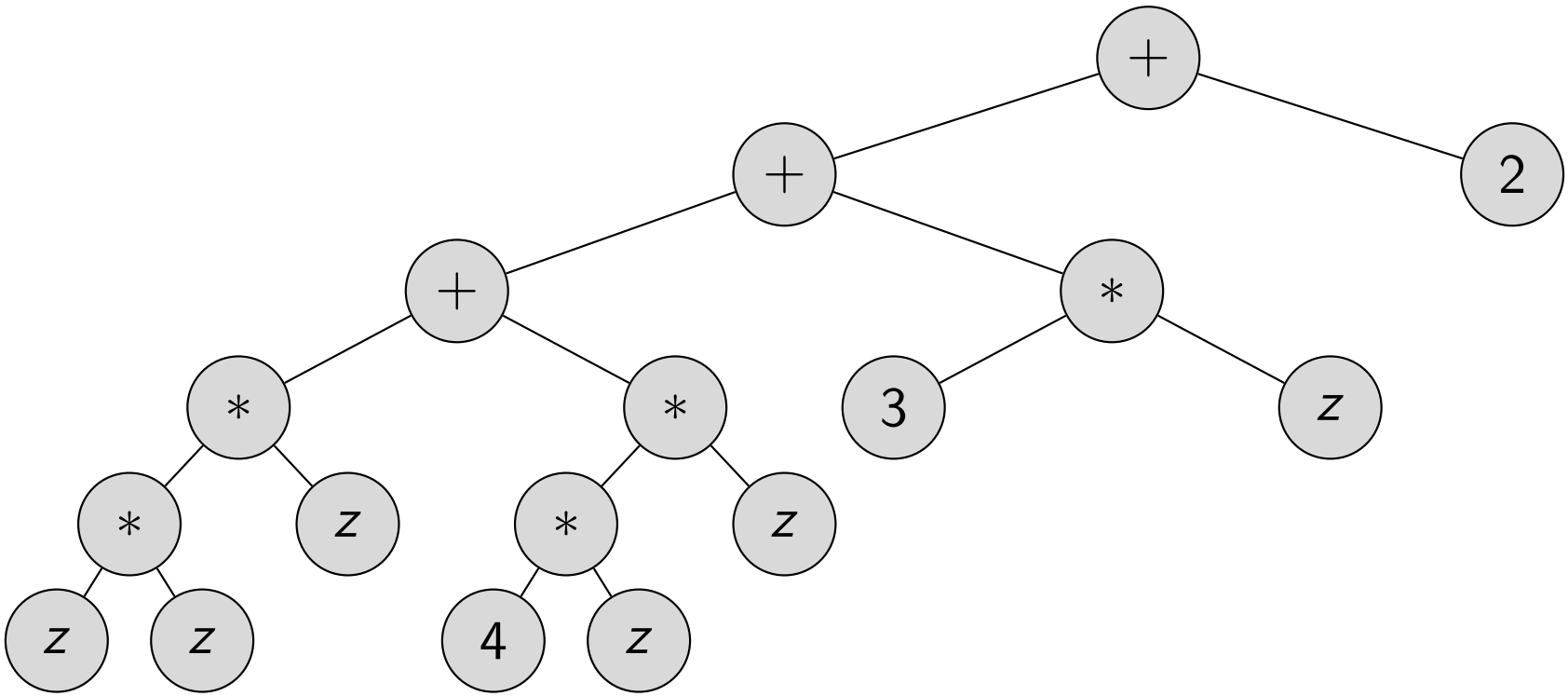



```
(assign (y) (- (+ (x)(* (y)(z))) (* (- (y)(z)) 3)))
```

Numeric Expressions

$$y := z^3 + 4z^2 + 3z + 2$$

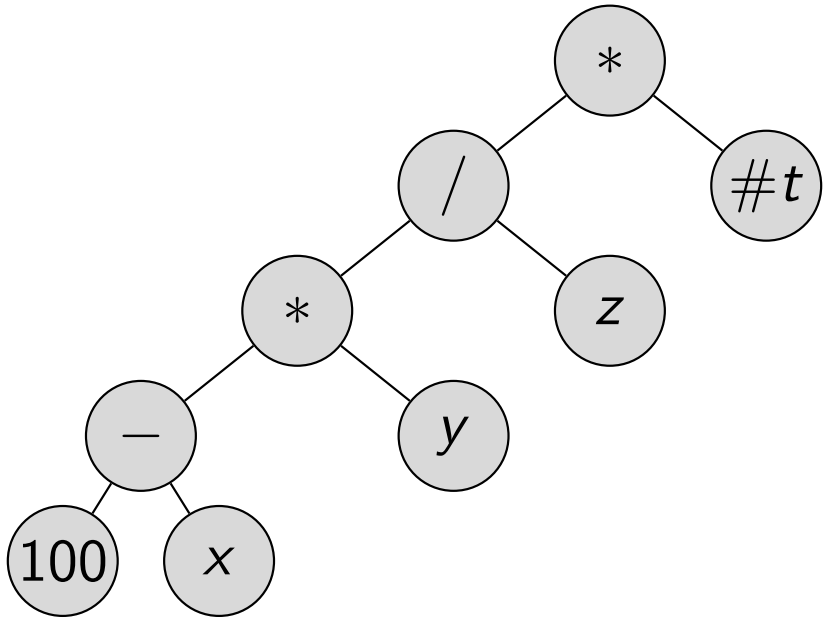
$$y := ((z * z * z + 4 * z * z) + 3 * z) + 2$$




```
(assign (y) (+(+(+(*(*(z)(z))(z))(*(*(4)(z))(z)))(* 3 (z))) 2 ))
```

Numeric Expressions

$$\frac{dx}{dt} = \frac{(100 - x) * y}{z}$$




```
(increase (x) (* (/ (* (- 100 (x))(y))(z)) #t)
```

Durative Action - Transformation

Transform the following durative action into a PDDL+ action-process-event

```
(:durative-action pour
:parameters (?a - bottle ?b - bottle)
:duration (= ?duration 10)
:condition (and
  (at start (> (litres ?a) 0))
  (at start (not (capped ?a)))
  (at start (not (capped ?b)))
  (over all (not (capped ?a)))
  (over all (not (capped ?b)))
)
:effect (and
  (at start (decrease (litres ?a) 1))
  (at end (increase (litres ?b) 1))
)
)
```

Durative Action - Transformation

Transform the following durative action into a PDDL+ action-process-event

```
  (:action pour-start
  :parameters (?a - bottle ?b - bottle)
  :precondition (and
    (> (litres ?a) 0)
    (not (capped ?a))
    (not (capped ?b))
  )
  :effect (and
    (assign (pour_clock ?a ?b) 0)
    (pour_started ?a ?b)
    (decrease (litres ?a) 1)
  )
)

(:process pour-process
  :parameters (?a - bottle ?b - bottle)
  :precondition (and
      (pour_started ?a ?b)
      (not (capped ?a))
      (not (capped ?b))
  )
  :effect (and
    (increase (pour_clock ?a ?b) #t)
  )
)
```

```
  (:event pour-end
  :parameters (?a - bottle ?b - bottle)
  :precondition (and
    (pour_started ?a ?b)
    (= (pour_clock ?a ?b) 10)
  )
  :effect (and
    (not (pour_started ?a ?b))
    (increase (litres ?b) 1)
  )
)

(:event pour-failure
  :parameters (?a - bottle ?b - bottle)
  :precondition (and
      (pour_started ?a ?b)
      (or (capped ?a)(capped ?b))
  )
  :effect (and
    (increase (pour_clock ?a ?b) 11)
  )
)
```

Modelling - Lifts I

In a building of 3 floors, there are 2 lifts and 6 people need to move between the floors. In the initial condition, the situation is

```
floor1 - lift1 p1 p2 p3
floor2 - p4 p5
floor3 - lift2 p6
```

at the end the situation must be

```
floor1 - p5 p6
floor2 - p1 p2
floor3 - p3 p4
```

The lifts can move *up* and *down*, and load and unload passengers. The capacity of each lift is infinite.
Model it with PDDL1.0

Modelling - Lifts I - domain.pddl

specificare variabili e azioni

```
(:types floor lift person)

(:predicates
    (at-person ?p - person ?f - floor) (at-lift ?l - lift ?f - floor)
    (on-top ?b - floor ?a - floor)(in ?p - person ?l - lift)
)

(:action up
    :parameters(?l - lift ?s - floor ?t - floor)
    :precondition(and (at-lift ?l ?s)(on-top ?t ?s))
    :effect(and (at-lift ?l ?t)(not (at-lift ?l ?s))))

(:action down
    :parameters(?l - lift ?s - floor ?t - floor)
    :precondition(and (at-lift ?l ?s)(on-top ?s ?t))
    :effect(and (at-lift ?l ?t)(not (at-lift ?l ?s))))

(:action load
    :parameters(?p - person ?l - lift ?f - floor)
    :precondition(and (at-lift ?l ?f)(at-person ?p ?f))
    :effect(and (not (at-person ?p ?f))(in ?p ?l)))

(:action unload
    :parameters(?p - person ?l - lift ?f - floor)
    :precondition(and (at-lift ?l ?f)(in ?p ?l))
    :effect(and (at-person ?p ?f)(not (in ?p ?l))))
```

Modelling - Lifts I - problem.pddl

```
(:objects
    l1 l2 - lift
    p1 p2 p3 p4 p5 p6 - person
    f1 f2 - floor
)

(:init
    (on-top f3 f2)(on-top f2 f1)
    (at-lift l1 f1)(at-lift l2 f3)
    (at-person p1 f1)(at-person p2 f1)(at-person p2 f1)
    (at-person p4 f2)(at-person p5 f2)
    (at-person p6 f3)
)

(:goal
    (at-person p5 f1)(at-person p6 f1)
    (at-person p1 f2)(at-person p2 f2)
    (at-person p3 f3)(at-person p4 f3)
)
```

Modelling - Lifts II

Change the domain such that the capacity of each lift is 1. Model it with PDDL1.0

Modelling - Lifts II - domain.pddl

```
(:types floor lift person)

(:predicates
    (at-person ?p - person ?f - floor) (at-lift ?l - lift ?f - floor)
    (on-top ?b - floor ?a - floor)(in ?p - person ?l - lift)
    (occupied ?l - lift)
)

(:action up
    :parameters(?l - lift ?s - floor ?t - floor)
    :precondition(and (at-lift ?l ?s)(on-top ?t ?s))
    :effect(and (at-lift ?l ?t)(not (at-lift ?l ?s))))

(:action down
    :parameters(?l - lift ?s - floor ?t - floor)
    :precondition(and (at-lift ?l ?s)(on-top ?s ?t))
    :effect(and (at-lift ?l ?t)(not (at-lift ?l ?s))))

(:action load
    :parameters(?p - person ?l - lift ?f - floor)
    :precondition(and (at-lift ?l ?f)(at-person ?p ?f)(not (occupied ?l)))
    :effect(and (not (at-person ?p ?f))(in ?p ?l)(occupied ?l)))

(:action unload
    :parameters(?p - person ?l - lift ?f - floor)
    :precondition(and (at-lift ?l ?f)(in ?p ?l))
    :effect(and (at-person ?p ?f)(not (in ?p ?l))(not (occupied ?l)))
```

Modelling - Lifts II - problem.pddl

```
(:objects
    l1 l2 - lift
    p1 p2 p3 p4 p5 p6 - person
    f1 f2 - floor
)

(:init
    (on-top f3 f2)(on-top f2 f1)
    (at-lift l1 f1)(at-lift l2 f3)
    (at-person p1 f1)(at-person p2 f1)(at-person p2 f1)
    (at-person p4 f2)(at-person p5 f2)
    (at-person p6 f3)
)

(:goal
    (at-person p5 f1)(at-person p6 f1)
    (at-person p1 f2)(at-person p2 f2)
    (at-person p3 f3)(at-person p4 f3)
)
```

Modelling - Lifts III

Now model it using numerical variables with PDDL2.1 LVL 2 and have a capacity of 2 for each lift.

Modelling - Lifts III - domain.pddl

```
(:types lift person)

(:predicates
    (in ?p - person ?l - lift))

(:functions
    (at-person ?p - person)
    (at-lift ?l - lift)
    (inside ?l - lift) (floors)
    (capacity)
)

(:action up
    :parameters(?l - lift)
    :precondition(and
        (< (at-lift ?l)(floors)
    )
    :effect(and (increase (at-lift ?l) 1)))

(:action down
    :parameters(?l - lift)
    :precondition(and (> (at-lift ?l) 1 )
    :effect(and (decrease (at-lift ?l) 1)))
```

```
(:action load
    :parameters(?p - person ?l - lift )
    :precondition(and
        (= (at-person ?p) (at-lift ?l))
        (< (inside ?l) (capacity)
    )
    :effect(and
        (in ?p ?l)
        (increase (inside ?l) 1)
    ))

(:action unload
    :parameters(?p - person ?l - lift)
    :precondition(and
        (in ?p ?l)
    )
    :effect(and
        (assign (at-person ?p) (at-lift ?l))
        (decrease (inside ?l) 1)
    ))
```

Modelling - Lifts III - problem.pddl

```
(:objects
    l1 l2 - lift
    p1 p2 p3 p4 p5 p6 - person
)

(:init
    (= (capacity 2))
    (= (floors 3))
    (= (at-lift l1) 1)(= (at-lift l1) 3)
    (= (at-person p1) 1) (= (at-person p2) 1) (= (at-person p3) 1)
    (= (at-person p4) 2) (= (at-person p5) 2)
    (= (at-person p6) 3)
)

(:goal
    (= (at-person p5) 1) (= (at-person p6) 1)
    (= (at-person p1) 2) (= (at-person p2) 2)
    (= (at-person p3) 3) (= (at-person p4) 3)
)
```

Modelling - Lifts IV

Now model it using durative actions in PDDL2.1 LVL 3 where lifts have duration $5tu$ when going up and $3tu$ when going down. Loading and unloading is instantaneous. Be careful to not let anybody in the lift while they are moving !

Modelling - Lifts V - domain.pddl

```
(:types lift person)

(:predicates
    (in ?p - person ?l - lift)
    (moving ?l - lift))

(:functions
    (at-person ?p - person)
    (at-lift ?l - lift)
    (inside ?l - lift) (floors)
    (capacity)
)

(:durative-action up
    :parameters(?l - lift)
    :duration(= ?duration 5)
    :condition(and
        (at-start (< (at-lift ?l)(floors))
    )
    :effect(and
        (at-start (moving ?l)
        (at-end (increase (at-lift ?l) 1))
        (at-end (not (moving ?l)))
    ))

(:durative-action down
    :parameters(?l - lift)
    :duration(= ?duration 3)
    :condition(and
        (at-start (> (at-lift ?l)1)
    )
    :effect(and
        (at-start (moving ?l)
        (at-end (decrease (at-lift ?l) 1))
        (at-end (not (moving ?l)))
    ))

(:action load
    :parameters(?p - person ?l - lift )
    :precondition(and
        (not (moving ?l))
        (= (at-person ?p) (at-lift ?l))
        (< (inside ?l) (capacity)
    )
    :effect(and
        (in ?p ?l)
        (increase (inside ?l) 1)
    ))

(:action unload
    :parameters(?p - person ?l - lift)
    :precondition(and
        (in ?p ?l) (not (moving ?l))
    )
    :effect(and
        (assign (at-person ?p) (at-lift ?l))
        (decrease (inside ?l) 1)
    ))
```