phiC

Compilers Project

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

CS3423: Compilers 2

Group18

Full Name	Enrollment No.
D Srinith	CS21BTECH11015
P Pranav Varma	CS21BTECH11044
M Santoshi Gayatri	CS21BTECH11036
K Akhila	CS21BTECH11031

Instructor: Prof. Jyothi Vedurada

Contents

1 | Introduction

1.1 | About Language

The DSL on the kinematics of point masses covers a range of essential topics in a concise and accessible manner. It begins with an exploration of basic equations of motion, including velocity, distance, and acceleration equations. Next, follows projectile motion, providing equations for trajectory calculation, range, time of flight, height, angle, and radius of curvature at any point. The documentation also addresses collisions, offering methods to calculate final velocities and analyze energy in cases of inelastic and elastic collisions, utilizing the coefficient of restitution and principles of linear momentum and energy conservation. This documentation serves as a comprehensive resource for understanding and applying key concepts in the kinematics of point masses.
Note: This language is designed specifically for working with point masses within a two-dimensional plane.

1.2 | Motivation

The motivation behind designing a DSL for the kinematics of point masses stems from the need to simplify and streamline the modeling and analysis of motion for educational and practical applications. It aims to simplify the learning process by providing an intuitive and user-friendly environment for exploring kinematics concepts.

1.3 | Uses and Goals

Simplicity: It that allows users, regardless of their programming background, to easily define and simulate the motion of point masses using a minimal amount of code.

Accuracy: The DSL's underlying equations and algorithms accurately represent the principles of kinematics, allowing for precise and reliable simulations and analyses.

Versatility: It supports a wide range of kinematic scenarios, including basic equations of motion, projectile motion, elastic and inelastic collisions, and the calculation of the center of mass.

2 | Features

2.1 | Datatypes

2.1.1 | Primitive Datatypes

Datatypes	Description
int	64-bit signed integer value
double	64-bit signed floating point value
string	sequence of characters in double quotes
bool	can be either true or false

Note: Characters are considered as string of length (1).

2.1.2 | Non Primitive Datatypes

Datatypes	Description
mass	mass of point object
time	measure of duration
position	location in 2D space (i,j)
velocity	measurement of the rate and direction of motion
acc	acceleration: rate of change of velocity
energy	the capacity of a system to do work
theta	measure of an angle
e	coefficient of restitution
distance	separation between two objects
momentum	the product of mass and its velocity

2.2 | Lexical Conventions

2.2.1 | Comments

Comments are non-executable text explanations within the code to provide clarity.

```
1) Single line comment - $

$ This line has been commented.

2) Multi Line Comment - $ ... $

$
   .
   .
   This Block of code has been commented
   ...
$
```

2.2.2 | Whitespace

Whitespace, tabs, indentations and new lines are ignored.

2.2.3 | Keywords

Keywords are specific words which are reserved and cannot be used as variables or for other purposes.

input	output	true	false
setr	addr	setv	addv
seta	adda	ke_after	pe_after
te_after	angle_after	get_traj	collide
time_to_collide	roc_after	p_after	s_after
time_to	loop	break	start
sin	cos	tan	mag

2.2.4 | Punctuators

Punctuator	Description
.	dot
:	colon
,	comma
”	double quote
’	single quote
{}	flower brackets
[]	square brackets
()	parenthesis

2.2.5 | Identifiers

An identifier is a name used to identify a variable, function, or other user-defined entity within a program. We followed C-style identifiers in this DSL.
Note: Reserved keywords and datatype names cannot be used as identifiers.

```
$ vari
$ _priVar
$ func_name1
$ var123
```

2.2.6 | Constants

Constants include integer, double, string and boolean values.

Datatype	Constant
int	9
double	6.747
string	”compilers”
bool	true

2.3 | Operators

Operators encompass symbols and keywords that manipulate data, including arithmetic, relational, logical, assignment operators.
Operators in their precedence order:

Operator	Description	Associativity
()	parenthesis	left to right
^	exponent	left to right
*	multiplication	left to right
/	division	left to right
%	modulo	left to right
+	addition	left to right
-	subtraction	left to right
==, !=	relational operators	left to right
<=, <, >=, >	relational operators	left to right
&	logical operator (AND)	left to right
|	logical operator (OR)	left to right
!	unary negation	right to left
++ , - -	unary operators	right to left
=	assignment operator	right to left

2.4 | Declarations

Declaration statements are used to define variables or functions, specifying their data type and name. By default variables are initialized to $\vec{0}$.

2.4.1 | Variable declaration

A variable is a named storage location that holds data. It is followed by a datatype and the statement ends by '.' - - [DOT]

```
    int x.
    double a, b = 0.1.
    velocity v = (5,10.7), u.
    position p = (-3,18.3).
    theta t = 45.
```

2.4.2 | Function declaration

Functions are blocks of code that can be called to perform specific tasks, encapsulating a sequence of instructions with optional input and output parameters.

```
  datatype func = (datatype x, datatype y) => {

    $ function body

  }
```

The function ”func” takes ”x” and ”y” as its arguments, where ”func” is the name of the function.

2.5 | Statements

Note: Expression statements, Jump statements, Call statement, Variable declaration statements should always end by ’ . ’ - - DOT

2.5.1 | Expression Statements

An expression statement is used for operations involving variable initialization, assignment, or modification.

Note that expression statement contains = (assignment operator) and the right-hand side (RHS) of this operator can contain a wide range of elements including values, vectors, booleans, function calls, and both unary and binary operators.

```
$ few examples:

x = 10.4.                   $ assigning a constant value
vel = (-3,10.1).            $ assigning a vector: -3i + 10.1j
valid = true.               $ assigning boolean
returnval = fn1(a,b).       $ return value of a function
z = x++.                    $ usage of unary operator
c = p * q.                  $ usage of Binary operator
```

2.5.2 | Predicate

Predicate is a term used to describe an expression that evaluates to either true or false. They are commonly used in decision control statements, loop statements and return statements. Some of the examples are shown below:

```
$ a < 10
$ age >= 18
$ (x == 5) | (y != 5)
$ b > 5 & b < 10
$ !n
$ !((x <= 3) & (y >= 9))
$ (age >= 18) & ((age != 19) | (age < 23))
```

2.5.3 | Decision Control Statements

Decision control statements allow us to control the flow of a program based on certain conditions. They help us to make decisions and execute different blocks of code depending on whether a given predicate is true or false.

```
If Syntax:

  <predicate>
  {
        $ Only IF BLOCK
  }

```

```
If - Else Syntax:

  <predicate>
  {
        $ IF BLOCK
  }
  {
        $ ELSE BLOCK
  }
```

2.5.4 | Jump Statements

Jump statements are used in programming to alter the normal flow of code execution. They allow you to jump to a different part of the program, based on conditions or specific requirements.

break : The 'break' keyword is employed to prematurely terminate the execution of a code block enclosed within a loop. Following the execution of the 'break' statement, the program's control flow proceeds to the line of the code immediately following the loop in which the 'break' statement is situated.

```
  .
  . . .
  .
    break.
  $ Exits the loop.
  ...
```

return : The 'return' keyword enables a function to exit prematurely, and can return a constant value of a specific data type, contingent upon the function's declared return type.

```
  .
  .
    => val.
  $ Exits the function with return value = val
  ..
  => .
  $ returns nothing
  => (k<l)
  ..
  $ returns boolean value of the predicate
  ..
  => "hi".
  $ returns string constant
  ...
```

2.5.5 | Loop Statements

Loop statements are commonly used for automating repetitive tasks and performing actions multiple times until a specific condition is met. They consist of a condition followed by a block of statements. These statements are repeatedly executed as long as the condition remains true before each iteration.

Note: Modifying the loop variable itself within the loop will not alter the original variable's

value in the broader scope.

```
loop <predicate>
{

  $ Code block

}
```

2.5.6 | Call Statements

Function call statement is used as an expression for RHS, that returns a value according to the function datatype.

```
$ say, defined function = int func(velocity v,time t).
$ function - func can be called as shown below:

dist = func(v,t).
```

2.6 | I/O Operations

input: Reading data from external source like keyboard.

```
$ Input syntax:

input: x.
input: x,y,z.
```

output: Used to display text to the console.

```
$ Output syntax:

output: "Compilers Project".
output: "Compilers Project title is " + project(x,y).
output: "Adding " + x + " and " + y + " gives " + add(x,y).
```

2.7 | Standard Libraries

Related to Magnitude

```
$ Returns magnitude of any vector v
Ex: velocity, position $

  mag(v)
```

Related to Position

```
$ Sets position of the point mass m1 as r1

(m1) setr (r1)
```

```
$ Updates position of the mass m1 by adding r1

(m1) addr (r1)
```

```
$ Returns position of mass m1 after time t1

(m1) r_after (t1)
```

Related to velocity

```
$ Sets velocity of the point mass m1 as v1

(m1) setv (v1)
```

```
$ Adds a velocity of v1 to the already
existing velocity of the point mass m1 $

(m1) addv (v1)
```

```
$ Returns the velocity of m1 after time t1
(m1) v_after (t1)

$ Returns the velocity of m1 after a distance s1
(m1) v_after (s1)

$ Based on the function arguments we are passing,
it selects the appropriate method to execute
  and returns the velocity $
```

Related to accerlation

```
$ Sets acceleration of the point mass m1 as a1

(m1) seta (a1)
```

```
$ New acceleration of mass m1 will be
equal to the current acceleration plus a1 $

(m1) adda (a1)
```

Related to Energy

```
$ Returns the Kinetic Energy of m1 after time t1

(m1) ke_after (t1)
```

```
$ Returns the Potential Energy of m1 after time t1

(m1) pe_after (t1)
```

```
$ Returns the Total Energy of m1 after time t1

(m1) te_after (t1)
```

Related to Angle

```
$ Returns the angle made by m1 after a time t1
(m1) angle_after (t1)

$ Returns the angle made by m1
after travelling a distance s1
(m1) angle_after (s1)

$ Based on the function arguments we are passing,
it selects the appropriate method to execute
and returns the angle with x-axis $
```

```
$ returns the trajectory of m1

trajectory t1 = get_traj(m1)
```

Related to Collision

```
$ Function to simulate a 2D collision
between two objects m1 and m2,
with a coefficient of restitution 'e'. $

$ If they are not at the same position,
it will print an error. $

$ m1 should be left of m2 at the time of collision.

$ The function updates properties such as
final velocities, momentum, etc of the masses.
```

```
  It returns loss of energy during collision. $

  (m1) collide (m2, e)
  (m1) collide (m2) $ takes e = 1 as default
```

```
  $ Returns the time it takes for them to collide
  if collision doesnt occur, it returns -1 $

  (m1) time_to_collide (m2)
```

Miscellaneous

```
  $ Returns total distance travelled by m1 in time t1

  (m1) s_after (t1)
```

```
  $ Returns the radius of curvature of m1 after time t1

  (m1) roc_after (t1)
```

```
  $ Returns the momentum of m1 after time t1

  (m1) p_after (t1)
```

```
  $ Returns time taken to go to the position r1

  (m1) time_to (r1)
```

```
  $ Returns time taken to acquire the velocity v1

(m1) time_to (v1)
```

3 | Example Codes

3.1 | Example Question 1

```
$
    Q) 2 objects A,B are seperated by distance 36m, are
    projected with equal velocities 10^(1/2) with angle 45
    and 135 anticlock. Find out several properties of objects
    before and after collision.

$


    start{

    point_mass m1 (10).
    point_mass m2 (15).
    time t.

    float vel_val.
    theta angle1,angle2.

    $ user provides values in terminal -> 45,135 (Given)
    input : angle1,angle2.
    input : vel_val.         $ Given, vel = 10^(1/2).
    (m1) setv (vel_val * sin(angle), vel_val * cos(angle)).
    (m2) setv (- vel_val * sin(angle), vel_val * cos(angle)).
    (m1) setp (-18,0).
    (m2) setp (18,0).

    t = (m1) time_to_collide (m2)

    <(t != 1)> {
        (m1) setv ((m1) v_after (t)).
        (m1) setv ((m1) r_after (t)).

        (m2) setv ((m2) v_after (t)).
        (m2) setv ((m2) r_after (t)).

        LE = (m1) collide (m2,e)
    }

    output: "Time taken to collide" + t.
    output: "Radius of curvature at 10s" + roc(10).
    output: "Distance travelled before collision"
            + (m1) r_after (t).
    output: "Angle at t = 10s" +  (m1) angle_after (10).
    output: "Energy loss during collision" + LE.

    }
```

3.2 | Example Question 2

```
$
    Q) (JEE ADVANCED 2022 P2) A projectile is fired from
    ground with velocity v at an angle theta from horizontal.
    If at the highest point the gravity changes from g to
    newg where newg = g/0.81 , If newd = n*d , where d is
    projectile range without change in gravity and newd is
    the range when above said thing is done , find the value
    of n.
$

    time find_time = (float k,int vel) => {
        return vel/k;
    }

    time find_time_2 = (float a ,int h) => {
        return (2*h/a)^(1/2)
    }

    start{
        mass m = 20.
        theta t = 45.
        position start = (0,0).
        acceleration g = (0,-10).
        (m) seta (g)
        (m) setv (10)

        flaot temp = ((m) v_after (0))->first.
        time max_point = find_time(10,temp).

        postition top = (m) getr (max_point).
        int distance_travelled = top->first.

        mass newm = 20
        (newm) setr (top).
        (newm) setv ((m) v_after (max_point)).
        $    g = g / 0.81.
            or this can be done
        $
        float a = g->second
        a = a / 0.81
        (m) seta ((0,a)).
        (newm) seta (g)

        int max_height_vertical = top->second
        max_point = find_time_2(a,max_height_vertical)

        positition final = (newm) r_after (max_point)

```

```
49          n = (final->first)/2*(distance_travelled).
50
51          output : "The value of n is " + n.
52      }
```