Assignment 0: Preparation
15-424/15-624/15-824 Foundations of Cyber-Physical Systems

Welcome to 15-424/15-624/15-824 Foundations of Cyber-Physical Systems!

This preparatory assignment is designed to help you prepare for the course. By working through these exercises you will begin to understand some of the basics underlying cyber-physical system (CPS) design.

When the proper assignments of the course are due, we will assume you have a good understanding of the necessary basic background. You will put it to good use to develop safe CPSs using the new techniques you will learn during the course. In order to give you a chance to remind yourself about the background we expect, this preparatory assignment 0 asks basic background questions.

It is OK if you cannot yet answer all of these questions now, as some will be easier and some harder. But *you should return the completed exercises to us when the course starts*, so that we can grade it for feedback, not for points. Each problem is marked with **easy**, **medium**, or **hard**; however, the level of difficulty for each question may depend upon your personal background. This preparatory assignment will help you identify which background areas you should devote especial attention to.

When you need to make assumptions for answering the questions, please write them down as part of your solution.

1 Math Background

Unsurprisingly, mathematical foundations play a big role not just in mathematical models for cyber-physical systems but also in their analysis.

1. **Derivatives:** Please compute the following derivatives.
   (a) $\left(5x^2+2\right)' = (5x^2)' + 2' = 5 \cdot 2x + 0 = 10x$
   (b) $\left(4x^3 + (5x)^2\right)' = \ (4x^3)' + \left((5x)^2\right)' = 12x^2 + 10x$
   (c) $\left((4x^2-2)(x^4+5)\right)' = \ (4x^2-2)'(x^4+5) + (4x^2-2)(x^4+5)'$
   $= 8x(x^4+5) + 4x^3(4x^2-2) = 8x^5 + 40x + 16x^5 - 8x = 24x^5 + 32$
   (d) (**medium**) $\left((4x-2)(x+5)^2\right)' =$

(e) (**medium**) $\left(\frac{4x^2-2}{x^4+5}\right)' =$

(f) (**medium**) $\cos(3x^2)' =$

2. **Integrals:** Can you solve the following indefinite integrals[1]?

   (a) $\int 5x^2+2dx = \int 5x^2dx + \int 2dx = 5\int x^2dx + 2x + C_2 = 5\frac{1}{3}x^3 + C_1 + 2x + C_2 = \frac{5}{3}x^3 + 2x + C$ where $C = C_1 + C_2$ is any constant of integration

   (b) (**easy**) $\int 4x^2 + xdx =$

   (c) (**easy**) $\int x^5 + 5x^3dx =$

   (d) (**easy**) $\int 2x^3 + (5x)^2dx =$

3. **Differential Equations**[2]

   An *initial value problem* (IVP) is a differential equation with an initial value assignment. The differential equation specifies how the variables evolve over time, and the initial value specifies where that trajectory starts at the initial time, say time 0.

   The variables evolve as a function of *time*, represented by an implicit variable t.

   (a)

$$\begin{bmatrix} x' & = & v \\ x(0) & = & x_0 \end{bmatrix}$$

   In this IVP, the derivative of x is given by v. Furthermore, we know that x's initial value, i.e. at time $t = 0$ is x_0. Thus, $x = x_0 + vt$ solves the differential

[1] To refresh your memory:
`http://www2.bc.cc.ca.us/resperic/math6a/lectures/ch5/1/anitderivatives.htm`

[2] If you need more serious reading material on differential equations, look for the book Ordinary Differential Equations by Tenenbaum and Pollard.

equation and the initial value, because we can plug it back into the differential equation and initial value condition to check

$$\begin{bmatrix} (x_0 + vt)' & = & 0 + v = v \\ (x_0 + v \cdot 0) & = & x_0 + 0 = x_0 \end{bmatrix}$$

(b) (**medium**) Notice that, now, v is not a constant like above, but itself also changes according to a.

$$\begin{bmatrix} x' & = & v \\ v' & = & a \\ x(0) & = & x_0 \\ v(0) & = & v_0 \end{bmatrix}$$

(c) (**hard**)

$$\begin{bmatrix} x' & = & -y \\ y' & = & x \\ x(0) & = & 0 \\ y(0) & = & 1 \end{bmatrix}$$

(d) (**hard**)

$$\begin{bmatrix} x' & = & x \cos t \\ x(0) & = & x_0 \end{bmatrix}$$

2 Physics Background

As the name hinted at, cyber-physical systems benefit from an understanding of basic physics.

1. Now that you are an astronaut trained in math, you've been sent to an alien planet (no, really)! Your task is to conduct the initial round of scientific experiments: The Bouncing Ball Tests$^{\text{TM}}$ on planet Zork.

   You will be dropping a ball from height H, with the ground of the planet at height 0. The position of the ball will be denoted by h (starting at $h = H$), and its velocity will be denoted by v. According to your spacesuit's sensors, the only force acting on the ball will be the planet's gravity g.[3]

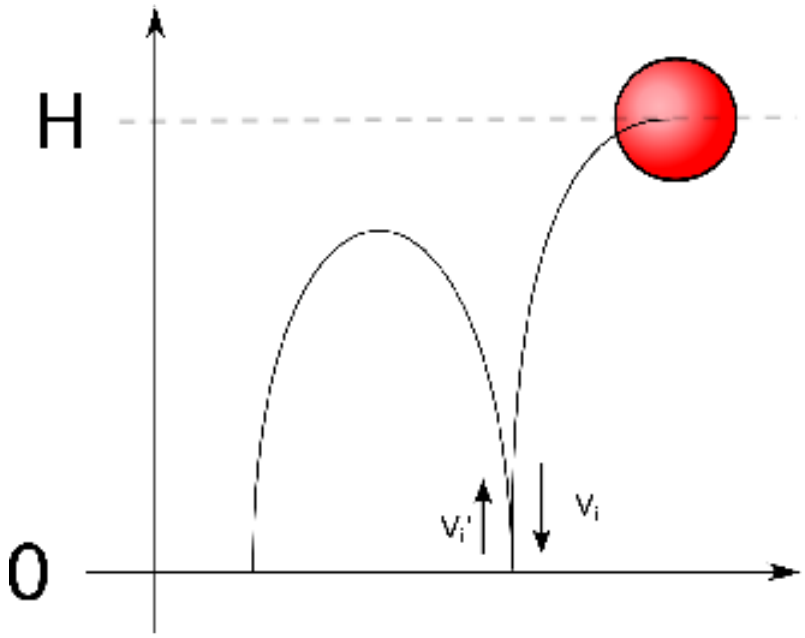
   (a) (**easy**) How long will the ball take to hit the ground if you drop it from height $h = H$?

   (b) (**easy**) Oops, you failed to measure the height H from which you dropped the ball! However, you noticed it took time τ to hit the ground. Can you figure out what height H the ball was dropped from?

   (c) (**medium**) What if instead of dropping the ball, you now throw it up with velocity v_0? What are the answers for the two above questions in that case?

[3] Refer to http://www.physicsclassroom.com/class/1DKin/Lesson-6/Kinematic-Equations for a refresher on the kinematic equations.

2. The ball just bounced! The surface of this planet is *uncanny*!



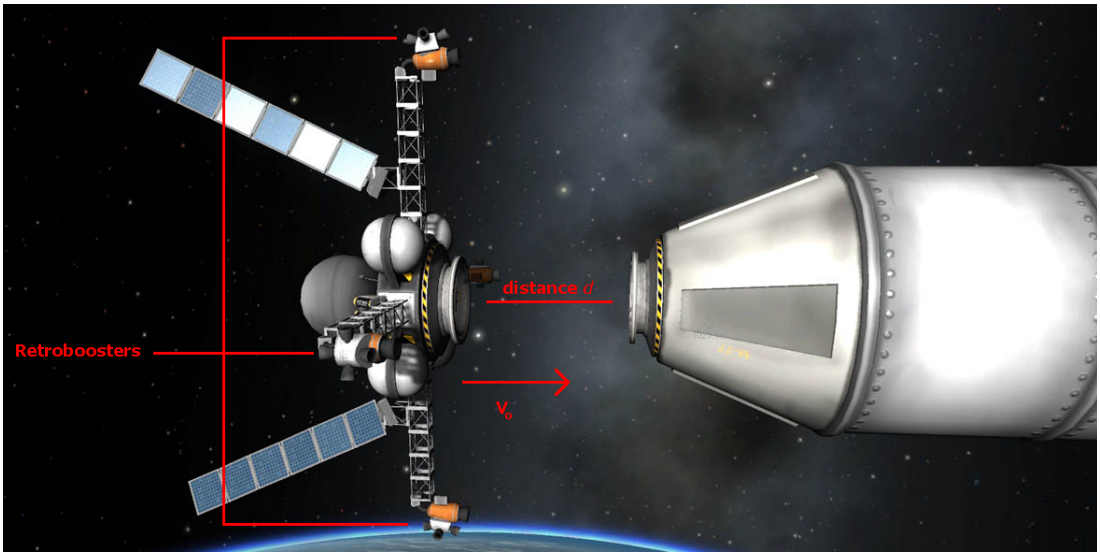
The picture above represents the scenario where you dropped the ball (on the right of the picture), which fell until it hit the ground with velocity V_i, bounced back up with velocity V_i^b, and finally came down again, hitting the ground a second time.

Unfortunately, you don't have sensors that can directly measure the velocities (don't ask, NASA budget cuts), so you don't know the exact values of V_i and V_i^b. You can, however, calculate them using a coefficient of restitution c, with $0 \leq c \leq 1$. The coefficient c reflects how much of the original impact velocity V_i the ball retains on the up-bounce velocity V_i^b.

To understand the bounciness characteristics of this alien planet, you're tasked to do the Bouncing Ball Tests$^{\text{TM}}$ again, this time by considering the *second* time the ball hits the ground:

(a) (**hard**) You drop the ball from height $h = H$ again. You already know when it will hit the ground the first time. Can you find out when it will hit the ground the second time, after it bounces?

3. Great! The experiments are done. You board your lander and get back into orbit.



Now you need to dock with the mothership, which will take you back to Earth. The mothership is stationary. Your lander is already perfectly aligned with it, at a distance of d. You are also heading towards the mothership with an initial velocity of v_0.

(a) (**easy**) How much *acceleration* a must your retro-boosters fire with so that you reach the mothership precisely when your velocity becomes 0, and thus perform a safe docking?

3 Logic Background

Cyber-physical systems also requires an understanding of basic logic for modeling and analysis purposes. We will start with first-order logic (FOL) for real arithmetic in this course. It is important that you familiarize yourself with it.

3.1 Propositional Connectives

In this section we will look at simpler formulas. These are composed of the following *logical connectives*:

$A \wedge B$	A and B	(conjunction)
$A \vee B$	A or B	(disjunction)
$A \rightarrow B$	A implies B	(implication)
$\neg A$	not A	(logical negation)

The connectives are used to put together simple arithmetical expressions, such as $x > 0$, that evaluate to true or false given values for their variables. You can even write formulas with polynomials in FOL: $x^2 + 5x + 3 > 0$. A logical formula is

- *valid* if it is true for all assignments, i.e., for all possible values that the variables could have,
- *satisfiable* if it is true for at least one assignment of variables, and
- *unsatisfiable* if it is not true for any assignment of variables.

In the following, determine if the statements are *valid*, *satisfiable*, **and/or** *unsatisfiable*.

(a) (**easy**) $2 < x \wedge x < 3$

Satisfiable, but not valid. We can find a value for x, say 2.5, that is greater than 2 and smaller than 3. This assignment *satisfies* both statements and hence their conjunction. However, we can find another value, like 4, which is greater than 2, but not smaller than 3, so that is *falsifies* one of the subformulas, and thus their conjunction. Hence, the formula is satisfiable but not valid.

(b) (**easy**) $3 < x \wedge x < 2$

(c) (**easy**) $x > 5 \vee x < 5$

(d) (**easy**) $x > 5 \vee x \leq 5$

(e) (**easy**) $x > 5 \wedge x \geq 5$

(f) (**medium**) $(x < y \wedge y < z) \rightarrow x < z$

(g) (**hard**) $(x > y \rightarrow x > z) \vee x > y$

(h) (**medium**) $x > y \leftrightarrow x^2 > y^2$

3.2 Quantifiers

Quantifiers[4] allow you write more expressive properties like "all birds fly". FOL for real arithmetic allows us to quantify specifically over the real numbers $\mathbb{R}$:

$\forall x\,(A(x))$ $A(x)$ is true "for every real number x".
$\exists x\,(A(x))$ $A(x)$ is true "for at least one real number x".

Let's look at some examples. The formula $\exists x\,(3 = 2 + 1 \wedge x = 5)$ is valid because we can find an x, namely 5, such that 3 is indeed $2 + 1$ (this is even true for *any* x, because x does not even occur), and x is also equal to 5 (which is certainly not true for *any* x, only for the particular choice 5 for x). The original assignment to x does not really matter here, since the quantifier "overrides" that value.

On the other hand, $\forall x\,(3 = 2 + 1 \wedge x = 5)$ would be *unsatisfiable* because the property does not hold for *every* real number. If we take $x = 0$, we have a counter-example. Even though $3 = 2 + 1$ is still true, $x = 5$ is not true and so, neither is the conjunction, $\wedge$.

Finally, what about $\exists x\,(x > y)$? This formula is *valid*, because no matter what value y has, there is always a number greater than y that we can choose for x to make $x > y$ true.

Again, determine if the statements are valid, satisfiable or unsatisfiable:

(a) (**medium**) $x < z \wedge \exists y\,(x < y \wedge y < z)$

(b) (**medium**) $\forall y\,(x < y)$

[4]Here's a quick read to refresh your mind about quantifiers: `http://cnx.org/content/m10728/latest/`

(c) (**hard**) $\forall x \, \exists y \, (x > y)$

(d) (**hard**) $\exists x \, \forall y \, (x > y)$

(e) (**medium**) $\exists x \, \exists y \, (x > y)$

(f) (**medium**) $\forall x \, \forall y \, (x > y)$

4 Program Contracts

Program contracts[5] are boolean expressions that describe properties of computer programs. Contracts are used to express safety and correctness properties of programs.

Throughout this course, we will build models of cyber-physical systems, develop safety and correctness contracts for these systems, and then prove that our models obey our contracts. This section introduces preconditions and postconditions in a simpler setting – integer functions written in a C-like programming language used in CMU's introductory computer science course Principles of Imperative Computation.

The most important contracts are preconditions and postconditions. Preconditions express properties that must hold prior to a program's execution and postconditions express properties that must hold after a program's execution. If a precondition of a program fails, it's the callers' fault, because he should not have called the program unless the input he provides meets the program's precondition. If a postcondition of a program fails, the program is to blame, because it promised to satisfy the postcondition on all inputs that meet its precondition. C0 is a C-like programming language that provides primitives for specifying preconditions and postconditions of functions.

The program below is a C0 program that computes the integer square root of `x`. The integer square root of a non-negative integer x is the greatest integer less than or equal to the square root of x. Because integer square roots are only defined for non-negative inputs, the `isqrt` function has a precondition that its argument is non-negative.

[5]Contracts are introduced in 15-122 Principles of Imperative Computation, a pre-requisite of this course. Masters or Ph.D. students who were waived from this requirement may want to read the relevant lecture notes from 15-122, which are available online: http://www.cs.cmu.edu/ fp/courses/15122-f15/lectures/01-contracts.pdf

```
int isqrt(int x)
//@requires(x >= 0);
{
  int c = 1;
  while(c * c <= x)
  {
    c = c + 1;
  }
  return c - 1;
}
```

Postconditions are specified in a similar way, except that the return value of the program is specified using `\result`. The program below is a C0 program whose specification states that the result of the computation must be at least as large as both inputs.

```
int max(int x, int y)
//@ensures( \result >= x && \result >= y && (\result == x || \result == y) );
{
  if(x >= y)
  {
    return x;
  }
  else
  {
    return y;
  }
}
```

(a) (**easy**) Identify a postcondition contract for the `isqrt` function defined above.

(b) (**easy**) Write a C function `int gcd(int x, int y)` that computes the greatest common divisor of two integers.

(c) (**easy**) Find a precondition and a postcondition for a function you wrote in part (b). The postcondition only needs to enforce that the return value of the function is a common divisor (the contract does not need to enforce that this divisor is the largest).

Hint. Use the following template for the last wo parts of this problem.

```
int gcd(int x, int y)
//@requires(______________________);
//@ensures(______________________);
{
  __(will span multiple lines)__
}
```

Don't forget to hand the assignment back to us by 1/15 :)