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

Welcome to 15-424/15-624/15-824 Logical 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. We have compiled relevant questions into this preparatory assignment in order to give you a chance to remind yourself about/familiarize yourself with the background we expect.

It is okay if you cannot yet answer all of these questions, 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 a number indicating roughly when in the course you will need to know the material; for example, you are expected to be comfortable with logic by *the end* of the second week, so those problems are marked **Week 2**. Material which you might not need for the course, but is potentially helpful if you have come across it is marked with a *. 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 special attention to when reviewing for this course.

When you need to make assumptions to answer a question, please write them down as part of your solution.

1 Math Background

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

1. **Derivatives:** Analyzing derivatives is one of the primary ways we reason about the differential equations that we use to model physical dynamics. When doing proofs on a computer, tools like our KeYmaera X prover can compute derivatives automatically, but understanding derivatives is essential to understanding your proofs, and as such you are expected to know how to compute them on written assignments and exams.

   Please compute the following derivatives (with respect to x).

   (a) $(5x^2 + 2)' = (5x^2)' + (2)' = 5 \cdot 2x + 0 = 10x$

(b) $(4x^3 + (5x)^2)' = (4x^3)' + ((5x)^2)' = 12x^2 + 50x$

(c) $\big((4x^2 - 2)(x^4 + 5)\big)' = (4x^2 - 2)'(x^4 + 5) + (4x^2 - 2)(x^4 + 5)'$
$= 8x(x^4 + 5) + 4x^3(4x^2 - 2)$
$= 8x^5 + 40x + 16x^5 - 8x^3$
$= 24x^5 - 8x^3 + 40x$

(d) (**Week 1**) $\big((4x - 2)(x + 5)^2\big)' =$

(e) (**Week 1**) $\left(\dfrac{4x^2 - 2}{5}\right)' =$

(f) (**Week 5**) $\left(\dfrac{4x^2 - 2}{x^4 + 5}\right)' =$

(g) (**Week 5**) $\big(\cos(3x^2)\big)' =$

2. **Integrals:** Understanding integrals helps us understand differential equations because solving differential equations is in many ways a generalization of integrating a function. In this course we will tackle differential equations by teaching you rigorous ways to prove properties about them without all the pain of actually solving them. That being said, knowing how to solve basic integrals will help you decide when it is or isn't easy to reason about a differential equation by looking at its solution and will help you understand why they have the solutions that they do.

   Can you solve the following indefinite integrals[1]?

[1] The way shown is but one of many possibilities, and certainly not the most effective one. We just chose it because it is reasonably systematic and reminds you of some laws of integration. To refresh your memory: `http://tutorial.math.lamar.edu/Classes/CalcI/ComputingIndefiniteIntegrals.aspx`

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

(b) (**Week 1**) $\int 4x^2 + x dx =$

(c) (**Week 1**) $\int x^5 + 5x^3 dx =$

(d) (*) $\int -6x \sin(3x^2) dx =$

3. **Ordinary Differential Equations**[2]

An *initial value problem* (IVP) is a system of differential equations together with an initial value assignment. The differential equations specify how the variables evolve over time, and the initial values specify 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)
$$\begin{pmatrix} x' & = & v \\ x(0) & = & x_0 \end{pmatrix}$$

We write IVPs as a system of equations in round parentheses, with one equation per line. Thus the notation above says that the derivative of x is given by v (at all times). Furthermore, we know that x's initial value, i.e. the value at time $t = 0$, is x_0. We see that $x = x_0 + vt$ solves the IVP, because we can plug this back into

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

the differential equation and initial value assignment to check:

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

Alternatively, this corresponds to integrating the function $x'(t) = v$ with constant of integration $C = x_0$, so $x(t) = x_0 + \int_0^t v ds = x_0 + vt$.

(b) (**Week 2**) Now, suppose v is not constant as above, but itself also changes according to a. Solve the following IVP:

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

(c) (**Week 5**) Here is an example that we would choose not to solve in a proof for this course, since the differential equations are themselves simpler to analyze than their solutions. For now, solve the following IVP:

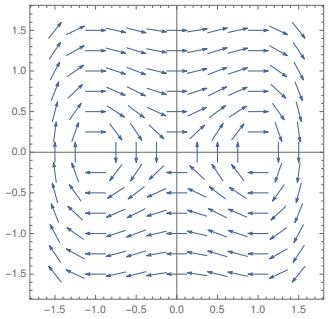
$$\begin{pmatrix} x' & = & -y \\ y' & = & x \\ x(0) & = & 0 \\ y(0) & = & 1 \end{pmatrix}$$

Notice that the differential equations we have encountered thus far do not mention the time variable t on their right-hand sides. These are also known as *autonomous* differential equations, because they do not depend on time. A quick way to obtain some qualitative properties of these systems is to sketch their direction fields.[3]

(d)

$$\begin{pmatrix} x' &= & y \\ y' &= & x - x^3 \end{pmatrix}$$

To sketch a direction field for this system, we draw at many points on the x, y plane directional arrows corresponding to the directions indicated by the right hand side. Intuitively, these indicate how trajectories of the system evolve at the points.



Really we should be drawing these vectors *at every point*, but then the picture would become too crowded. Still, even a sparsely drawn vector field can provide useful qualitative information. From the sketch above, we guess, for example, that there is clockwise circular motion close to the points at $x = \pm 1, y = 0$.

(e) (**Week 5**) Sketch a direction field for the following system. How does the sketch relate to your solution to the IVP involving the system above (3c)?

$$\begin{pmatrix} x' &= & -y \\ y' &= & x \end{pmatrix}$$

[3]For a nice visualization of 2D systems, check out: `https://anvaka.github.io/fieldplay/`
If you know Mathematica, use its `StreamPlot` function.

(f) (**Week 5**) Sketch a direction field for the following system. Briefly indicate some qualitative properties (e.g. asymptotic behavior) of the system from the sketch.

$$\begin{pmatrix} x' &= & y \\ y' &= & 2x \end{pmatrix}$$

2 Physics Background

As the name already gave away, cyber-physical systems benefit from an understanding of basic physics. In this question, you will remind yourself about the intuition of simple physical processes that will be of relevance in the course.

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 $x_0 > 0$, with the ground of the planet at height 0. The position of the ball will be denoted by x (starting at $x = x_0$), 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, with acceleration due to gravity denoted by g.[4] You may assume that x, v, g are all given in standard units, i.e. m, ms^{-1}, ms^{-2} respectively.

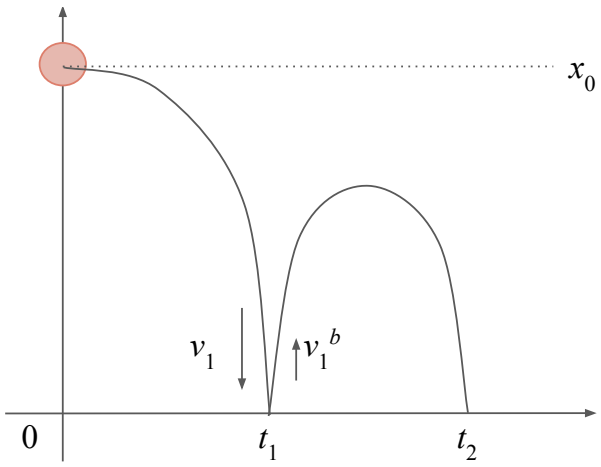
   (a) (**Week 2**) How long will the ball take to hit the ground if you drop it from height $x = x_0$?

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

(b) (**Week 2**) Oops, you failed to measure the height x_0 from which you dropped the ball! However, you noticed it took time t_1 (measured in seconds) to hit the ground. Can you retroactively figure out what height x_0 the ball was dropped from?

(c) (*) 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?

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_1, bounced back up with velocity v_1^b, and finally came down again, hitting the ground a second time.

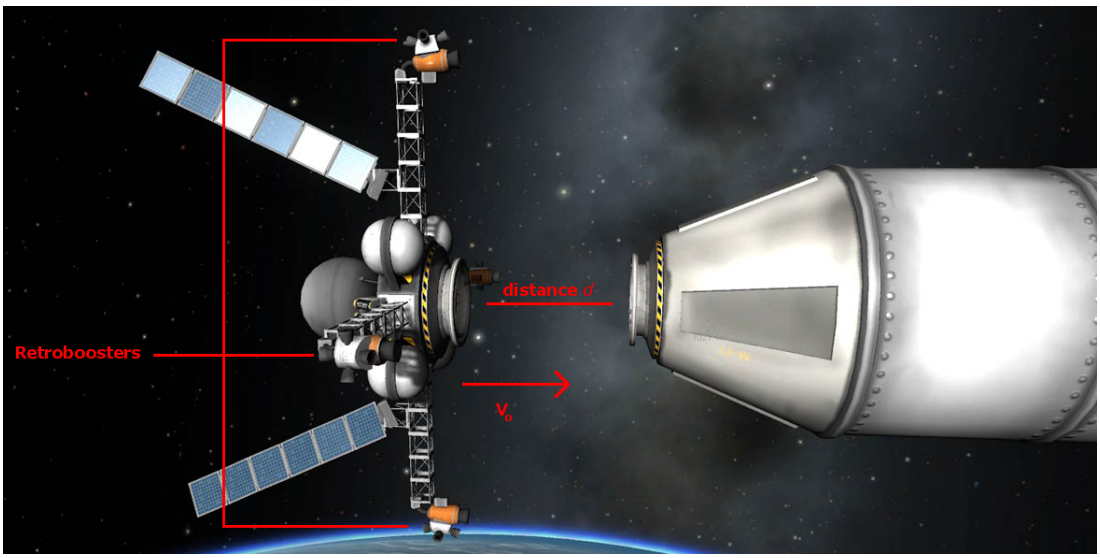
Unfortunately, you don't have sensors that can directly measure the velocities (don't ask, NASA budget cuts), so you don't know the exact values of v_1 and v_1^b. You can, however, calculate them using a coefficient of restitution c, with $0 \leq c \leq 1$. The coefficient c describes how much of the original impact velocity v_1 the ball retains when it bounces up, i.e. :

$$v_1^b = cv_1$$

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

(a) (*) You drop the ball from height $x = x_0$ again. You already know when it will hit the ground the first time (at $t = t_1$). Can you find out when it will hit the ground the second time (at $t = t_2$), 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 heading towards the mothership with an initial velocity of v_0.

(a) (**Week 3**) 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 require an understanding of basic logic for modeling and analysis purposes. We will start with first-order logic (FOL) for *real arithmetic* in this course, which provides boolean operators (like and/or) and also arithmetic comparisons. Many students find logic to be the most difficult topic in the course, so do not be alarmed if you find some of these problems difficult. However, you should take this opportunity to familiarize yourself with logic, because you will be expected to become fluent in logic in the second week. This course uses logic both to describe relevant aspects of physical systems and to reason about cyber-physical systems.[5]

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)
$A \to B$	A implies B	(implication)
$\neg A$	not A	(logical negation)

The connectives are used to put together simple arithmetical comparisons, such as $x > 0$, that evaluate to true or false given values for their variables (which range over the real numbers). You can even write formulas with polynomials in FOL, such as $x^2 + 5x + 3 > 0$, which will be true or false depending on the value of variable x. A logical formula is:

- *valid* if it is true for all possible values that the variables could have. For example, $x^2 \geq 0$ is valid, because it will evaluate to true for every *real* value of x.
- *satisfiable* if it is true for at least one value of its variables. For example, $x > 0$ is satisfiable, because there are real values for x, such as 5, that satisfy $x > 0$.
- *unsatisfiable* if it is not true for any assignment of variables. For example, $x > 0 \wedge x < 0$ is unsatisfiable, because no value for x will make both conjuncts true simultaneously.

Determine if the following formulas are *valid*, *satisfiable*, **and/or** *unsatisfiable*. Briefly justify your answers.

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

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.

[5] A course on Logic and Proof is available at: `http://leanprover.github.io/logic_and_proof`.

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) (**Week 1**) $3 < x \wedge x < 2$

(c) (**Week 1**) $x > 5 \vee x < 5$

(d) (**Week 1**) $x > 5 \vee x \leq 5$

(e) (**Week 1**) $\neg(x > 5 \wedge x \leq 5)$

(f) (**Week 2**) $(x < y \wedge y < z) \rightarrow x < z$

(g) (**Week 2**) $x > y \leftrightarrow x^2 > y^2$

(h) (**Week 2**) $(x > y \rightarrow x > z) \vee x > y$

Besides building formulas out of atomic inequalities, we may also build propositional formulas out of atomic propositions. For example, suppose the propositional symbol A^{15}_{122} means "Has taken 15-122" and A^{21}_{120} means "Has taken 21-120", then the formula $A^{15}_{122} \wedge A^{21}_{120}$ means "Has taken both 15-122 and 21-120".

(i) (**Week 2**) Writing requirements as logical formulas allows us to avoid (informal) descriptions in English that may be vague or ambiguous. Using the notation above, write a propositional formula ϕ that describes the prerequisites for this course. The prerequisites are reproduced below; for simplicity, we have removed the "or equivalent" clauses:

- 15-122 Principles of Imperative Computation
- 21-120 Differential and Integral Calculus
- and (21-241 Matrix algebra or 15-251 Great Theoretical Ideas in Computer Science or 18-202 Mathematical Foundations of Electrical Engineering)
- Substitutes: 21-242 Matrix Theory or 21-341 Linear Algebra I for 21-241

(j) (**Week 2**) Write a formula ψ that describes the courses that you have taken out of the ones listed in the course requirements. You may pretend to have taken a course if you took equivalent courses, e.g. graduate students may list A_{122}^{15} in ψ.

(k) (**Week 2**) What is the (logical) relationship between ϕ and ψ that represents the fact that you have satisfied the course requirements?

3.2 Quantifiers

Quantifiers[6] allow us to write more expressive properties like "all birds fly". FOL for real arithmetic allows us to quantify specifically over the real numbers $\mathbb{R}$:

$\forall x\,(A(x))$ means $A(x)$ is true "for every real number x".
$\exists x\,(A(x))$ means $A(x)$ 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 in the formula $3 = 2 + 1$), and x is also equal to 5 (which is certainly not true for *any* x, only for the particular choice of 5).

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

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.

Determine if the following formulas are *valid*, *satisfiable*, **and/or** *unsatisfiable*. Briefly justify your answers.

(a) (**Week 2**) $\forall y\,(x < y)$

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

(b) (**Week 2**) $\exists x \, \exists y \, (x > y)$

(c) (**Week 2**) $\forall x \, \forall y \, (x > y)$

(d) (**Week 2**) $\forall x \, \exists y \, (x > y)$

(e) (**Week 2**) $\exists x \, \forall y \, (x > y)$

(f) (**Week 2**) $x < z \rightarrow \exists y \, (x < y \wedge y < z)$

4 Program Contracts

Program contracts[7] are boolean expressions that describe properties of computer programs. In this course, we will use contracts to express safety and correctness properties of cyber-physical system programs, then prove that those properties hold. It is helpful but not required to have some experience reasoning about correctness with contracts. This section introduces contracts 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, called C0. Please note that *we never use C0 in this course* and C0 itself is not a prerequisite. But it is a good opportunity for you to exercise program contracts.

The most important contracts are preconditions and postconditions. Preconditions express expectations that must hold prior to a program's execution and postconditions express guarantees that must hold after a program's execution. If a precondition of a program fails, it's the caller's fault, because they should only be calling the program with inputs that meets the program's precondition. If a postcondition of a program fails, however, 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 mechanisms 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.

[7] 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 the variable `\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, and that it is equal to either one of the 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) (**Week 2**) Identify a postcondition contract for the `isqrt` function defined above.

(b) (**Week 2**) Write a C function `int gcd(int x, int y)` that computes the greatest common divisor of two integers.

(c) (**Week 2**) Find a precondition and a postcondition for the function you wrote in (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 two 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 08/30 :)