

Assignment 1: Introduction to Hybrid Programs
15-424/15-624/15-824 Logical Foundations of Cyber-Physical Systems

TAs: Irene Li (mengzeli@andrew.cmu.edu)

Yong Kiam Tan (yongkiat@cs.cmu.edu)

Due Date: Thursday, September 13th, 11:59PM (no late days), worth 60 points

1. **Terms, formulas, hybrid programs, oh my!** For each of the following, determine if the expression is a (syntactically) well-formed **dL** term, a well-formed **dL** formula, a well-formed hybrid program, or none of the above (i.e., it is not well-formed). In the case that the expression is none of the above, give a short explanation why.

(a) $z := x^5 - 1$

(b) $?(x \cdot y \cdot z > \frac{3}{4})$

(c) x

(d) $42 + 6 \cdot 9$

(e) $[g := 42]$

(f) $z + 1 := x^5$

(g) $x := y + 1 \cup; x = y'$

2. **Operator precedence.** Adopting a set of operator precedence rules helps reduce the number of parentheses (or braces) needed when writing down an expression. However, it is *essential* that you are familiar with these rules to avoid hours of debugging/misunderstanding in your labs/theory assignments!

For convenience, here is a cheatsheet for the operator precedence rules in **dL**. Refer back here whenever you are not sure about how to parse a given expression.

- In the theory assignments and the textbook, parentheses (\cdot) are used to disambiguate terms, formulas and programs. For clarity, we will always write braces around differential equations like $\{x' = v, v' = a\}$ and $\{t' = 1 \ \& \ t \leq T\}$.

In KeYmaera X (and in your lab assignments), parentheses (\cdot) are used for terms and formulas, but braces $\{\cdot\}$ are used to group programs.

- Unary operators always bind stronger than binary operators. This **includes the first-order and modal quantifiers**. Examples:

– $\forall x P \wedge Q \equiv (\forall x P) \wedge Q$ (similarly for $\exists x$),

– $[\alpha]P \wedge Q \equiv ([\alpha]P) \wedge Q$ (similarly for $\langle \alpha \rangle$),

– $\neg P \wedge Q \equiv (\neg P) \wedge Q$,

– $\alpha; \beta^* \equiv \alpha; (\beta^*)$.

- The arithmetic operators have their usual precedence from mathematics.

- The binary logical connective \wedge binds stronger than \vee , which in turn binds stronger than $\rightarrow, \leftrightarrow$. To avoid confusion, there is no default binding precedence between \rightarrow and \leftrightarrow . Explicit disambiguating parentheses are required when these appear in sequence. Examples:

$$- P \wedge Q \vee R \equiv (P \wedge Q) \vee R$$

- $P \rightarrow Q \leftrightarrow R$ is considered illegal, and must be disambiguated either as $(P \rightarrow Q) \leftrightarrow R$ or $P \rightarrow (Q \leftrightarrow R)$.

- Hybrid program operator $;$ binds tighter than \cup . Example:

$$- \alpha; \beta \cup \gamma \equiv \{\alpha; \beta\} \cup \gamma$$

- All arithmetic operators $+, -, \cdot$ associate to the left. All logical and program operators associate to the right. In particular, implication (\rightarrow) associates to the right. Examples:

$$- a - b - c \equiv (a - b) - c$$

$$- P \rightarrow Q \rightarrow R \equiv P \rightarrow (Q \rightarrow R).$$

$$- \alpha; \beta; \gamma \equiv \alpha; (\beta; \gamma).$$

Although many of these operators satisfy an associativity law (e.g., $a + (b + c) = (a + b) + c$), it is important to know their default associativity because that is also how KeYmaera X parses expressions.

For this question, you will practice applying the above precedence rules. For each formula/program below, add parentheses/braces indicating the correct binding for the connectives.

(a) $[y := 5]x = 3 \vee x = 5 \rightarrow x + 1 = 6$

(b) $\exists x x = 5 \rightarrow x + 1 = 6 \rightarrow x = 1$

(c) $[x := 5; y := y + x \cup \{x' = v, v' = a \& v = -1 \vee v = 1 \wedge v = 2\}]x > 0$

3. **Evolve nondeterministically!** This question will test your understanding of nondeterministic evolution.

$$\beta \stackrel{\text{def}}{\equiv} x := x_0; v := v_0; t := 0; \{x' = v, v' = a, t' = 1 \& v \geq 0\}; ?v = 0$$

Intuitively, hybrid program β first sets the initial values of x, v to x_0, v_0 , and the initial value of the clock variable t to 0. It then runs the differential equations (where a is a constant acceleration) subject to the evolution domain constraint $v \geq 0$. Finally, it tests that $v = 0$ at the end of the run.

- (a) Assume that $a < 0 \wedge v_0 \geq 0$. At the end of a run of hybrid program β , what is the value of t as a function of x_0, v_0 , and a ?

Let us modify our program a little by removing the test:

$$\gamma \stackrel{\text{def}}{=} x := x_0; v := v_0; t := 0; \{x' = v, v' = a, t' = 1 \& v \geq 0\}$$

- (b) Again assuming that $a < 0 \wedge v_0 \geq 0$, what are the possible values of v at the end of a run of γ ? What about the possible values of t ?
- (c) Suppose we assume instead that $a < 0 \wedge v_0 \leq 0$ (v_0 is **less than** or equal to zero). What are the possible values of v and t at the end of a run of β ?
- (d) Let us consider some **dL** formulas that use the above programs β and γ . For each of the following formulas, state whether the formula is valid and give a brief explanation why. (The antecedents correspond to the various sign assumptions on a and v_0 from the previous parts of this question.)
 - i. $a < 0 \wedge v_0 \geq 0 \rightarrow [\beta]v = 0$
 - ii. $a < 0 \wedge v_0 < 0 \rightarrow [\beta]v = 0$
 - iii. $a < 0 \wedge v_0 < 0 \rightarrow \langle \beta \rangle v = 0$
 - iv. $a < 0 \wedge v_0 \geq 0 \rightarrow [\gamma]v = 0$
 - v. $a < 0 \wedge v_0 \geq 0 \rightarrow \langle \gamma \rangle v = 0$

Hint: Carefully review the semantics of differential equations with evolution domain constraints $\{x' = f(x) \& Q\}$.

4. **Search for the truth!** Determine whether each of the following formulas is valid/satisfiable/unsatisfiable. If the formula is satisfiable, describe the set of states in which it is satisfiable. If it is unsatisfiable, briefly explain why.

- (a) $\forall x \langle \{x' = c\} \rangle x > 0$
- (b) $[?x \geq 0; x := -x] x < 0$
- (c) $\langle \{z' = -c \& z > 0\}; \{z' = c \& z < 0\} \rangle z = k$

5. **Find a program!**

- (a) Write down a program α that makes the formula $\forall z (x = z \rightarrow [\alpha]x > z)$ valid.
- (b) Write down a program α that makes the following formula satisfiable, but not valid: $[\alpha]z > 5$
- (c) Write down a program α that makes the formula $\forall x \forall y \langle \alpha \rangle x = y$ valid. The program may mention x **but not** y .

6. **Define an operator!** The primitive operators of hybrid programs can be used to define more complex operators. For example, the if-then-else statements can be defined as follows:

$$\mathbf{if } Q \mathbf{ then } \alpha \mathbf{ else } \beta \stackrel{\text{def}}{=} ?Q; \alpha \cup ?\neg Q; \beta$$

- (a) Define the n -ary switch statement which runs program α_i if formula P_i is true and chooses nondeterministically if multiple conditions are true:

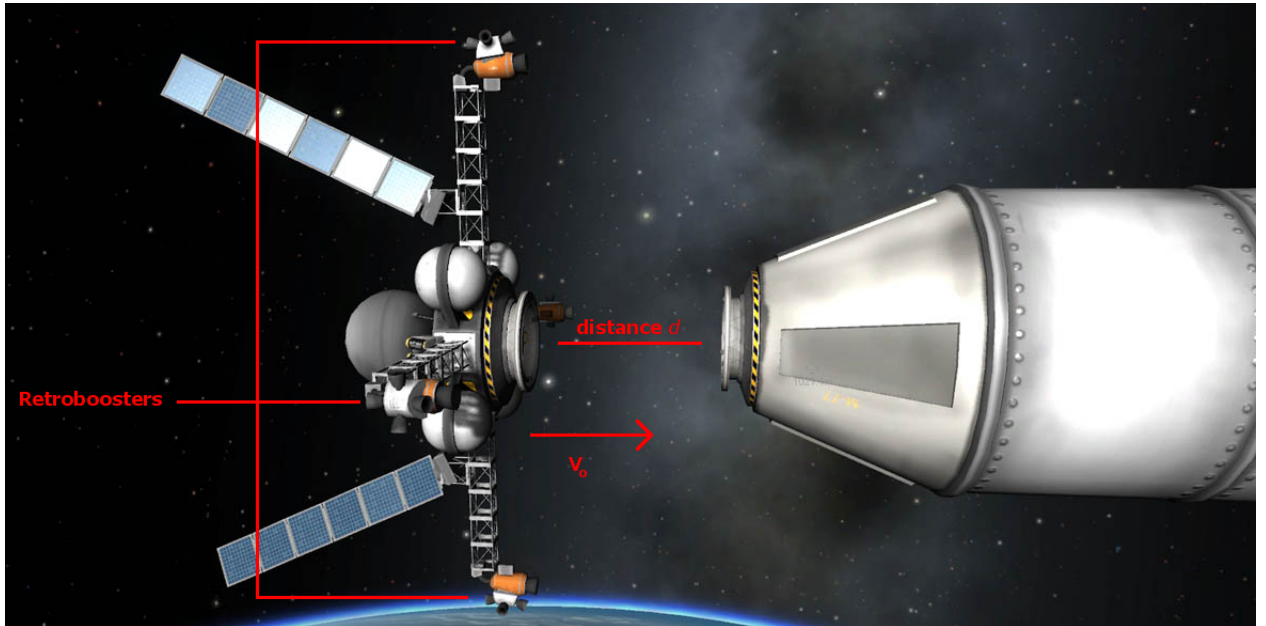
```
switch {  
  case  $P_1 : \alpha_1$   
  case  $P_2 : \alpha_2$   
  :  
  case  $P_n : \alpha_n$   
}
```

- (b) Suppose we added a catch-all case to the switch with a fallback program β :

```
switch {  
  :  
  case  $_ : \beta$   
}
```

Change your definition so that only α_i corresponding to the **first** true condition P_i executes. If none of the P_i 's are true, then execute the fallback program β .

7. **Build a model in SPACE!** In Theory 0 we considered the problem of docking a lander with a stationary mothership.



Recall that the lander and mothership are perfectly aligned, they are initially separated by distance d , and the lander is already moving toward the mothership with non-zero initial velocity v_0 .

In Theory 0, you already computed the constant acceleration that the retro-boosters on the lander should fire with so that the lander's velocity reaches 0 precisely when it reaches the mothership.

- (a) Write a hybrid program to model the situation assuming that the lander is accelerating at rate a_0 . A template for the hybrid program is given below:

```
( --- ; --- ; --- );          /* Assign initial values to variables */
{ --- , --- , --- & --- }    /* Run the continuous dynamics subject
                             to a domain constraint */
```

Hint: Remember to model the lander *braking to a stop*.

- (b) A *safety* property is something that a cyber-physical system should *always* maintain. Write down a **dL** formula that expresses the safety property that the lander does not collide with the mothership.
- (c) We also want to make sure that our cyber-physical system is efficient. In this case, it means that the lander should not stop before it reaches the mothership (we don't want to be stuck in outer space :/). Write down a **dL** formula that expresses this property.
- (d) Let's put together the program and the properties into a **dL** formula we can later verify in KeYmaera X. You will also need to add some initial assumptions (requirements) on the known variables to make the formula valid.

Don't worry if you don't know how to verify this yet, we will be going over that in Lab 1. For now, fill in the following template:

```
( --- )      ->      /* Initial assumptions (requirements) */
[ { ----- } ]     /* Hybrid program (from part a) */
( --- )      /* Safety and efficiency conditions
              (from parts b and c) */
```