IFT 6755 -H2019 | PROJECT 2: PREDICATE LOGIC

The project's due date is Friday March 22nd, 24:00 AoE. Submit your solutions on StudiUM as a ZIP file containing all relevant PDFs, text, source files and Alloy instance screenshots.

Provide source files with the full plain-text listings for Z3 or Alloy. Your ZIP file should also contain a README.txt describing exactly where I can find the answer for each part.

For Part 1, it is best if you typeset your answers with LaTeX. If you choose to do the exercises by hand, make sure your handwriting is very clear (no cursive!) and submit a legible scan of your solution.

Try to provide Alloy theme files that help the readability of your results.

The 4 parts of the project add up to 105%.

PART 1 (30%)

Solve the following exercises from the book (Huth & Ryan, Logic in Computer Science, Second Edition):

1. Page 157, Exercise 2.1.1
2. Page 157, Exercise 2.1.3
3. Page 160, Exercise 2.2.4
4. Page 161, Exercises 2.3.9 k, l, m
5. Page 163, Exercise 2.4.1
6. Page 163, Exercise 2.4.6

PART 2 (30%)

Using Alloy (version 4.2 greater), solve the following exercises from the book (Huth & Ryan, Logic in Computer Science, Second Edition):

1. Page 166, Exercise 2.7.2
2. Page 167, Exercise 2.7.4
3. Page 167, Exercise 2.7.5

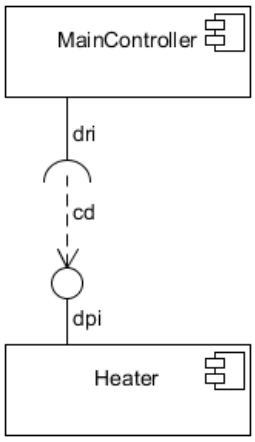
PART 3 (5%)

One way to use Z3 as predicate logic solver is using quantifiers and the theory of uninterpreted functions. Here is a proof of the validity of $\vDash \forall x P(x) \rightarrow \exists x P(x)$:

```
; let A be an arbitrary domain
(declare-sort A)

; let P be an arbitrary predicate on the atoms in A
; (note that we are using an uninterpreted boolean function to define a predicate!)
(declare-fun P (A) Bool)

(define-fun conjecture () Bool
  (=>
    (forall ((x A)) (P x))
    (exists ((x A)) (P x))
  )
)

; if our conjecture is valid, its negation is unsat
(assert (not conjecture))
(check-sat)
```

Use Z3 to complete Exercises 2.3.9 (k), (o), (r) in page 161 of the book.

PART 4 (40%)



We can use UML to describe component architectures. For example, in the diagram to the left, we show the washing machine **W1**, with two components, *MainController* and *Heater*. The *MainController* component *requires* some functionality, represented by the *dri* socket symbol. The *Heater* component *provides* some functionality via an interface represented by the *dpi* ball symbol. In this washing machine, these two components have been paired via the *cd* delegation link.

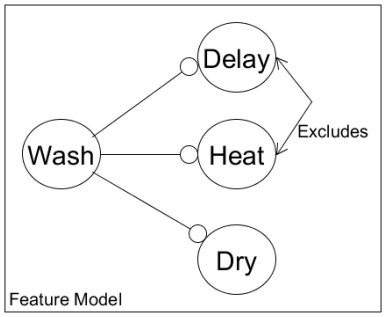
Step 1. Using Alloy (version 4.2 greater), define a specification of component diagrams, where each component can have any number of socket or ball ports that can be connected via directed links (socket→ball). A component may not delegate functionality to itself.

Step 2. A common pattern for representing indivudual instances in Alloy is by defining singleton extensions, as in the snippet below:

```
abstract sig Feature {}
one sig Heat, Delay, Dry extends Feature {}
```

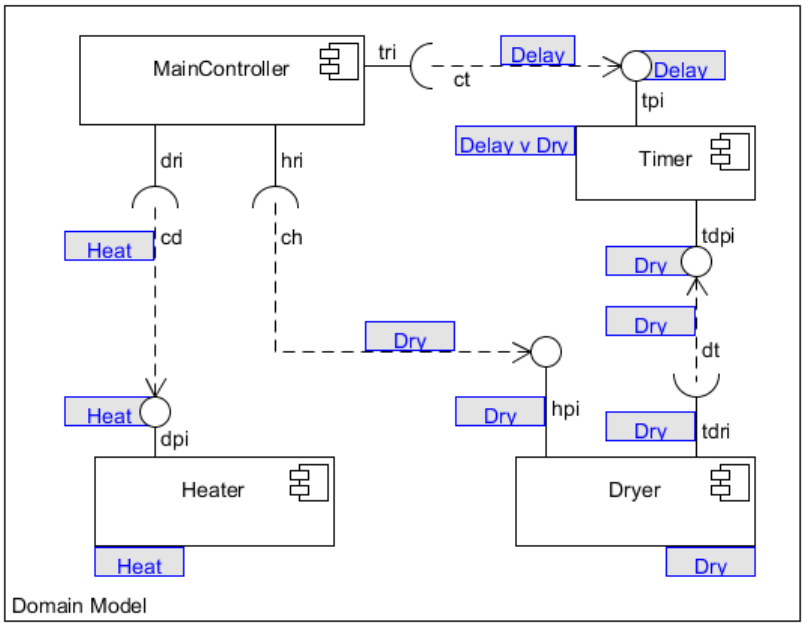
Use this definition pattern (albeit for your signatures for components, ports, etc.) to represent the washing machine **W1** by extending your spec from Step 1.

Step 3. Read Step 4. Then encode in Alloy the SPL **W** of washing machine controller components:



Show that **W1** can be derived from **W** for the configuration {*Heat=T, Dry=F, Delay=F*}.

Then show that nothing else other than **W1** can be derived from **W** for the configuration {*Heat=T, Dry=F, Delay=F*}.



Step 4. Write a property to verify that in every product all required functionalities are delegated to some provided interface. Your property should be in terms of the signatures defined in Step 1. In other words, it should not be hard-coded for this specific product line; instead it should be applicable to any product line of component diagrams.