Logical Formalism
Homework

Jim Newton and Adrien Pommellet, EPITA

September 17, 2024

Your assignment should comply with these mandatory rules:

- All your answers must be written in a single file named `folo.py` based on the provided template file.
- Your Python script should be devoid of syntax errors or any mistake that would interfere with its execution. If this is not the case, the entire assignment will be ignored. Test your code on a Linux distribution by executing the command line `python3 folo.py` in a directory that features the right dependencies, without relying on an IDE.
- Please use the exact function names / signatures stated in this document (and as given in the provided files).
- You can run the provided test file `folo_test.py` in the same directory as your answer file in order to perform a rough check that will return failed assertions (or nothing if your code is working fine).
- Test every function you write on one or more examples.
- Be wary of dependent functions: a wrong answer to an early question may compromise further results.

Not respecting these rules may prevent the automatic grading system from reading your work, resulting in a grade of 0 for the current assignment that can't be challenged.

1 About Python

1.1 Basic Python

Python is an interpreted programming language; Python source files use the `.py` extension. The instruction `python3` *`filename.py`* invokes the interpreter on the source file `filename.py`: the instructions (excluding definitions) in the file's body are then executed in the order they were written. If no input file is provided, the interpreter is run in interactive mode and can be closed by pressing `Ctrl+D`. We will favour writing our code in dedicated files over using this interactive mode.

This assignment can be completed using almost exclusively straightforward instructions on sets and `for` loops shown in the following example:

```
# Defines a set.
es = {0, 1, 2}
# Add a single element to a set.
es.add(3)
# Defines an empty set.
# ems = {} does not work as intended, be careful.
ems = set()
# Iterates on the elements of a set.
# Prints "0 1 2 3".
for x in es:
  print(x)
# We can also use the keyword 'in' as a membership predicate.
# Prints "one".
if 1 in es:
  print("one")
# Two variables can simultaneously iterate on sets of pairs.
pairs = {(0, 1), (1, 2), (2, 0)}
# Prints 1 3 2.
for (x, y) in pairs:
  print(x + y)
# Returns AssertionError if a proposition is not true.
assert(1 + 1 == 2)
assert(2 + 2 == 2)
# A function can return a function defined in its body.
def f(k):
  def h(x):
    return (x+k)
  return h
g = f(3)
# Prints 5.
print g(2)
```

1.2 Quantifiers in Python

Python features an implementation of the universal and existential quantifiers $\forall$ and $\exists$ thanks to the keywords `any` and `all` that can be used in combination with predicates (Boolean functions) and iterators on sets. Practically speaking, you can test property of the form $\forall x \in E,\ P(x)$, $\forall x, y \in E,\ P(x, y)$, or $\forall x, y \in E,\ q(x, y) \implies p(x, y)$ as follows:

```
def even(n):
  return (n % 2 == 0)
e = {2, 6, 10, 11}
# Determines whether all the elements of e are even.
# Prints False.
print(all(even(n) for n in e))
# Determines whether all the elements of e smaller than or equal
# to 10 are even.
# Prints True.
print(all(even(n) for n in e if (n <= 10)))
f = {-2, -1, 3}
# Determines whether the sum of an element of e and an element
# of f is always positive.
# Prints false.
print(all((n + m > 0) for n in e for n in f))
```

The instruction `any` can be used in a similar fashion:

```
     # Determines whether at least one element of e is a
     # multiple of 5.
     # Prints True.
print(any((n % 5 == 0) for n in e))
```

2 Functions

For the rest of this document, we consider that E and F are two sets. Let $\sim$ be a binary relation on $E \times F$. It is said to be:

- a *partial function* if, given $x \in E$ and $y_1, y_2 \in F$, $\big((x \sim y_1) \wedge (x \sim y_2)\big) \implies (y_1 = y_2)$. Intuitively, not every element of E admits an image, but if it does, then this image is unique.
- a *function* if, given $x \in E$, $\exists! y \in F$, $x \sim y$.

Question 1. Implement a Python function `is_relation` that takes a finite subset `es` of E, a finite subset `fs` of F, and a finite set `pairs` of pairs in $E \times F$ as input and returns `True` if and only `pairs` defines a binary relation on `es` $\times$ `fs`.

Question 2. Implement a Python function `is_partial_function` that takes a finite subset `es` of E, a finite subset `fs` of F, and a finite set `pairs` of pairs in $E \times F$ as input and returns `True` if and only `pairs` defines a partial function `es` $\to$ `fs`. Don't forget to check that `pairs` defines a binary relation on `es` $\times$ `fs` beforehand.

Question 3. Implement a Python function `is_function` that takes a finite subset `es` of E, a finite subset `fs` of F, and a finite set `pairs` of pairs in $E \times F$ as input and returns `True` if and only `pairs` defines a function `es` $\rightarrow$ `fs`. Don't forget to check that `pairs` defines a partial function `es` $\rightarrow$ `fs` beforehand.

3 Bijections

Question 4. Implement a Python function `is_injection` that takes a Python function (not a set of pairs representing a binary relation, unlike the previous section) `f` of type $E \rightarrow F$, a finite subset `es` of E, and a finite subset `fs` of F as input and returns `True` if and only f is an injection `es` $\rightarrow$ `fs`.

Question 5. Implement a Python function `is_surjection` that takes a Python function `f` of type $E \rightarrow F$, a finite subset `es` of E, and a finite subset `fs` of F as input and returns `True` if and only f is a surjection `es` $\rightarrow$ `fs`.

Question 6. Implement a Python function `is_bijection` that takes a Python function `f` of type $E \rightarrow F$, a finite subset `es` of E, and a finite subset `fs` of F as input and returns `True` if and only f is a bijection `es` $\rightarrow$ `fs`.

Question 7. Implement a Python function `find_inverse` that takes a Python function `f` of type $E \rightarrow F$, a finite subset `es` of E, and a finite subset `fs` of F as input and returns a function `h` of type $F \rightarrow E$ that is the inverse of `f` if `f` is a bijection `es` $\rightarrow$ `fs`, and `None` otherwise.

4 Relations

Let $\sim$ be a binary relation on $E \times E$. It said to be:

- *symmetric* if, $\forall x, y \in E$, $(x \sim y) \implies (y \sim x)$.
- *antisymmetric* if, $\forall x, y \in E$, $\big((x \sim y) \land (y \sim x)\big) \implies (x = y)$.
- *reflexive* if, $\forall x \in E$, $x \sim x$.
- *transitive* if, $\forall x, y, z \in E$, $\big((x \sim y) \land (y \sim z)\big) \implies (x \sim z)$.

Question 8. Implement a Python function `is_symmetric` that takes a finite subset `es` of E and a finite set `pairs` of pairs in $E \times E$ as input and returns `True` if and only `pairs` defines a symmetric relation on `es` $\times$ `es`. Don't forget to check that `pairs` defines a binary relation on `es` $\times$ `es` beforehand.

Question 9. Implement a Python function `is_antisymmetric` that takes a finite subset `es` of E and a finite set `pairs` of pairs in $E \times E$ as input and returns `True` if and only `pairs` defines an antisymmetric relation on `es`$\times$`es`. Don't forget to check that `pairs` defines a binary relation on `es` $\times$ `es` beforehand.

Question 10. Implement a Python function `is_reflexive` that takes a finite subset `es` of E and a finite set `pairs` of pairs in $E \times E$ as input and returns `True` if and only `pairs` defines a reflexive relation on `es` $\times$ `es`. Don't forget to check that `pairs` defines a binary relation on `es` $\times$ `es` beforehand.

Question 11. Implement a Python function `is_transitive` that takes a finite subset `es` of E and a finite set `pairs` of pairs in $E \times E$ as input and returns `True` if and only `pairs` defines a transitive relation on `es` $\times$ `es`. Don't forget to check that `pairs` defines a binary relation on `es` $\times$ `es` beforehand.

Warning: you might be tempted to solve this exercise by writing three concentric loops, (i.e., n^3). However, doing so will probably result in test failures because of time-out issues. Try using Python predicates of the form `if (x in e)` to improve the algorithm's runtime.

5 Equivalence and Order Relations

An *equivalence relation* $\sim$ on $E \times E$ is a reflexive, symmetric, and transitive binary relation. Given $x \in E$, the *equivalence class* of x according to $\sim$ is the set $[x]_\equiv = \{y \in E \mid x \equiv y\}$ of all elements equivalent to x.

A *partial order relation* $\sim$ is a reflexive, antisymmetric binary relation. It is said to be *total* if $\forall x, y \in E$, $x \sim y$ or $y \sim x$: intuitively, two elements can always be compared.

Question 12. Implement a Python function `is_equivalence` that takes a finite subset `es` of E and a finite set `pairs` of pairs in $E \times E$ as input and returns `True` if and only `pairs` defines an equivalence relation on `es` $\times$ `es`. Don't forget to check that `pairs` defines a binary relation on `es` $\times$ `es` beforehand.

Question 13. Implement a Python function `gen_equiv_class` that takes a finite subset `es` of E, an element `e` of E, and a finite set `pairs` of pairs in $E \times E$ and outputs the set of all elements of E equivalent to `e` according to `pairs`. Don't forget to check that `pairs` defines an equivalence relation on `es` $\times$ `es` beforehand and that `e` is an element of `es`, using the `assert` Python instruction.

Question 14. Implement a Python function `is_partial_order` that takes a finite subset `es` of E and a finite set `pairs` of pairs in $E \times E$ as input and returns `True` if and only `pairs` defines an order relation on `es` $\times$ `es`. Don't forget to check that `pairs` defines a binary relation on `es` $\times$ `es` beforehand.

Question 15. Implement a Python function `is_total_order` that takes a finite subset `es` of E and a finite set `pairs` of pairs in $E \times E$ and returns `True` if and only `pairs` defines a total relation on `es` $\times$ `es`. Don't forget to check that `pairs` defines an order relation on `es` $\times$ `es` beforehand.