FACULTY OF COMPUTER SCIENCE IAȘI

@ FII

UNIVERSITATEA „ALEXANDRU IOAN CUZA“ din IAȘI

Lab 2

- Back to main

Main information

In this lab, you will use Dafny to model and verify properties of digital circuits.

Digital circuits are built from simple components called logic gates that perform basic operations on binary signals (0 or 1, or equivalently, false or true). By combining these gates, we can build complex computational systems.

A simple circuit that computes (A AND B) OR (NOT C):

```
A ──┐
    ├─ AND ─┐
B ──┘       │
            ├─ OR ── Output
C ── NOT ───┘
```

Circuit data type

1. (0.5p) Define a algebraic datatype Circuit in Dafny that represents digital circuits. Your datatype should include:

- Input pins (you may want to declare a separate enumerative type for it)
- The basic gates: not, and, or

```
datatype Input = // ...
datatype Circuit =
// ...
```

Evaluation

Now we need a way to evaluate a circuit given specific values for the input pins.

2. (0.5p) Define a function `eval` that takes a `c: Circuit` and `inputs: Input -> bool` and outputs `bool`.

```
function eval(c: Circuit, inputs: Input -> bool): bool
{
  //...
}
```

3. (0.5p) Consider the following functions:

```
function sum_bit_fn(i1: Input, i2: Input): Circuit
{
    And(Or(Input(I1), Input(I2)), Not(And(Input(I1),Input(I2))))
}
function carry_bit_fn(i1: Input, i2: Input): Circuit
{
    And(Input(I1), Input(I2))
}
```

Write a function that computes the sum of two bits, using the definitions above. The signature of the function is:

```
function halfAdder(i1 : Input, i2 :Input, inputs: Input -> bool) : (bool, bool)
{
    // ...
}
```

Note that the function returns a pair of two booleans, where the first component is the sum of the two bits and the second component is the carry.

4. (0.5p) Write four tests that illustrate the `halfAdder` works correctly.

Proofs

5. (1p) Write a predicate `equiv` that captures the equivalence of two circuits. Using this predicate, prove the following equivalences

Law	Equivalence
Excluded Middle	A ∨ ¬A ≡ ⊤ (always true)
De Morgan 1	¬(A ∨ B) ≡ ¬A ∧ ¬B
De Morgan 2	¬(A ∧ B) ≡ ¬A ∨ ¬B

Law	Equivalence
Distributive 1	A ∨ (B ∧ C) ≡ (A ∨ B) ∧ (A ∨ C)
Distributive 2	A ∧ (B ∨ C) ≡ (A ∧ B) ∨ (A ∧ C)

6. (1p) Write a function for implication. An example is (you might have a different implementation, depending on your type definitions):

Law	Equivalence
Double Negation Elimination	¬¬A → A
Contrapositive	(A → B) → (¬B → ¬A)
Peirce’s Law	((P → Q) → P) → P
Łukasiewicz Axiom 1	A → (B → A)
Łukasiewicz Axiom 2	(P → (Q → R)) → ((P → Q) → (P → R))
Łukasiewicz Axiom 3	(¬P → ¬Q) → (Q → P)