1 Programmation déclarative

La famille en Prolog

```
% Des faits, séparés par des points
child(john,sue).       child(john,sam).
child(jane,sue).       child(jane,sam).
child(sue,george).     child(sue,gina).
male(sam).             male(george).
female(sue).           female(jane).       female(june).

% Des règles, également séparées par des points
parent(Y,X)  :- child(X,Y).
father(Y,X)  :- child(X,Y), male(Y).
grand_father(X,Z) :- father(X,Y), parent(Y,Z).
opp_sex(X,Y) :- male(X), female(Y).
opp_sex(Y,X) :- male(X), female(Y).

% Des requêtes (ou objectifs), et leurs réponses
?- father(george, sue)
   true.
?- grand_father(george, X)
   X = john ; X = jane. % tous les X vérifiant l'expression
```

Recipients en TLA$^+$

```
(*****************************************)
(* Getting 4 liters from a 5 l and a 3 l jugs  *)
(*****************************************)

EXTENDS Naturals \* we use numbers

VARIABLES big,   \* amount of liters in big jug
          small  \* amount of liters in small one

TypeInvariant == /\ small \in 0..3
                 /\ big   \in 0..5

(** Initial state **)
Init == big = 0 /\ small = 0

(** Possible transitions **)
FillSmallJug  ==  small' = 3     /\ big' = big
FillBigJug    ==  small' = small /\ big' = 5
EmptySmallJug ==  small' = 0     /\ big' = big
EmptyBigJug   ==  small' = small /\ big' = 0

Min(m,n) == IF m < n THEN m ELSE n

SmallToBig == /\ big'   = Min(big + small, 5)
              /\ small' = small - (big' - big)
BigToSmall == /\ small' = Min(big + small, 3)
              /\ big'   = big - (small' - small)

(** Next-state relation **)
Next ==  \/ FillSmallJug
         \/ FillBigJug
         \/ EmptySmallJug
         \/ EmptyBigJug
         \/ SmallToBig
         \/ BigToSmall

(** Complete specification **)
Spec == Init /\ [][Next]_<<big, small>>

(* Property of which we want a counter-example *)
NotSolved == big # 4
```

2 OOP vs. FP : A cat catches a birds and eats it

Version OOP

```
class Bird;
class Cat {
  void catch(Bird b);
  void eat();
private:
  Bird prey;
}

int main() {
  auto cat = new Cat();
  auto bird = new Bird();
  cat.catch(bird);
  cat.eat();
}
```

Version FP

```
class Bird;
class Cat;
class Catcher;
class CatReplete;

Catcher    catch(Cat c, Bird b);
CatReplete eat(Catcher c);

int main() {
  std::cout <<  eat( catch(Cat(), Bird()) );
}
```