Rapport TouIST "Water Sort Puzzle"

Tristan Salvan, Raffael Dwyyana, Corentin Pereira, Manolo Sardo

Decembre 2022

Introduction

Le projet consiste à modéliser différentes situations d'un jeu à l'aide de nombreuses conditions prédéfinies. Le principe du jeu est le suivant : plusieurs tubes remplis ou non de différentes couleurs sont présents, le but est d'arriver à un état pour lequel chaque tube contient une seule couleur ou est vide, pour ce faire, il est possible de verser une couleur d'un tube à un autre, une fois par étape, du moment que la couleur versée soit similaire à la couleur sur laquelle est versée (s'il y en a une).

Pour modéliser ce jeu, nous utilisons TouIST comme langage pour la logique propositionnelle. Nous utilisons également un afficheur écrit en Python par l'un de nos membres pour nous aider à visualiser l'état de chaque étape. Nous préférons utiliser TouIST en ligne de commande au lieu de l'interface graphique car il donne une réponse plus rapide, sachant que nous faisons beaucoup de tests. Pour faciliter l'écriture du code, au lieu d'utiliser l'IDE officiel de TouIST, nous tapons tout le code dans Visual Studio Code tout en se servant de l'extension TouIST.

1 Modélisation des conditions

1.1 Exercice 1

La situation initiale (État 0)

Nous avons commencé par modéliser la situation initiale (état 0) de la Figure 1 en utilisant la norme de modélisation de l'énoncé pour y appliquer les conditions.

```
$C=[vert, orange, bleu, rouge, rose, vide]
$T=3
$k=0
$ETATS=[0..$k]

p(0, 1, 1, vert) ;; the first tube, first layer is green
p(0, 1, 2, orange) ;; the first tube, second layer is orange
p(0, 1, 3, vert) ;; the first tube, third layer is green
p(0, 1, 4, orange) ;; the first tube, fourth layer is orange

p(0, 2, 1, orange) ;; the second tube, first layer is orange
p(0, 2, 2, vert) ;; the second tube, second layer is green
p(0, 2, 3, orange) ;; the second tube, third layer is orange
p(0, 2, 4, vert) ;; the second tube, fourth layer is green

p(0, 3, 1, vide) ;; the third tube, first layer is empty
p(0, 3, 2, vide) ;; the third tube, second layer is empty
p(0, 3, 3, vide) ;; the third tube, third layer is empty
p(0, 3, 4, vide) ;; the third tube, fourth layer is empty
```

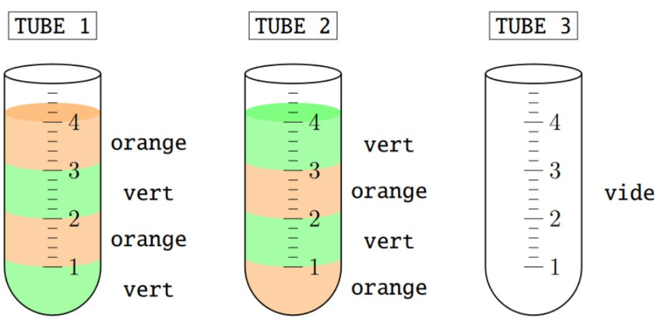

FIGURE 1 – Exercice 1

1.1.1 Les règles

Les premières conditions (Exercice 1) n'ont pas posé de grand problème à modéliser, en s'aidant de ce que l'on a vu en tp nous avons pu modéliser les 3 premières conditions sans trop de soucis.

```
;; first rule
bigand $i in $ETATS: ;; for each state
  bigand $t in [1..$T]: ;; for each tube
    bigand $e in [1..4]: ;; for each layer
      bigor $c in $C: ;; at least one color
        p($i, $t, $e, $c)
        ;; every state, every tube, every layer,
        ;; have one color at minimum
      end
    end
  end
end

;; second rule
bigand $i in $ETATS: ;; for each state
  bigand $t in [1..$T]: ;; for each tube
    bigand $e in [1..4]: ;; for each layer
      bigand $c in $C: ;; for each color
        bigand $c2 in $C when $c2 != $c:
        ;; for each color different from the previous one
          not p($i, $t, $e, $c) or not p($i, $t, $e, $c2)
          ;; if a state, a tube, a layer have a color,
          ;; it can't have another color
        end
      end
    end
  end
end

;; third rule
bigand $i in $ETATS: ;; for each state
  bigand $t in [1..$T]: ;; for each tube
    bigand $e in [1..3]: ;; for each layer
      p($i, $t, $e, vide) => p($i, $t, $e+1, vide)
      ;; if a state, a tube, a layer is empty,
      ;; then the layer above it is empty as well
    end
  end
end
```

Les résultats sont :

$$\bigwedge_{i \in ETATS} \bigwedge_{t \in [1..T]} \bigwedge_{e \in [1..4]} \bigvee_{c \in C} P_{i,t,e,c}$$

$$\bigwedge_{i \in ETATS} \bigwedge_{t \in [1..T]} \bigwedge_{e \in [1..4]} \bigwedge_{c \in C} \bigwedge_{\substack{c2 \in C, \\ c2 \neq c}} \neg P_{i,t,e,c} \vee \neg P_{i,t,e,c2}$$

$$\bigwedge_{i \in ETATS} \bigwedge_{t \in [1..T]} \bigwedge_{e \in [1..3]} P_{i,t,e,vide} \Rightarrow P_{i,t,e+1,vide}$$

1.2 Exercice 2

La situation initiale (État 0)

Pour cet exercice, nous commençons également par initialiser le premier état du jeu.

```
$C=[vert, orange, bleu, rouge, rose, vide]
$T=3 ;; nombre de tubes
$k=8
$ETAPES=[0..$k]

p(0, 1, 1, vert) ;; the first tube, first layer is green
p(0, 1, 2, orange) ;; the first tube, second layer is orange
p(0, 1, 3, vert) ;; the first tube, third layer is green
p(0, 1, 4, orange) ;; the first tube, fourth layer is orange

p(0, 2, 1, orange) ;; the second tube, first layer is orange
p(0, 2, 2, vert) ;; the second tube, second layer is green
p(0, 2, 3, orange) ;; the second tube, third layer is orange
p(0, 2, 4, vert) ;; the second tube, fourth layer is green

p(0, 3, 1, vide) ;; the third tube, first layer is empty
p(0, 3, 2, vide) ;; the third tube, second layer is empty
p(0, 3, 3, vide) ;; the third tube, third layer is empty
p(0, 3, 4, vide) ;; the third tube, fourth layer is empty
```

1.2.1 Règle 1

Pour la première règle, nous nous assurons que chaque tube est soit vide, soit rempli d'une seule couleur. Pour cela nous nous sommes inspirés de la modélisation de la carte avec les couleurs vue en cours.

```
bigand $t in [1..$T]: ;; for each tube
  bigor $c in $C: ;; at least one color
    bigand $e in [1..4]: ;; for each layer
      p($k, $t, $e, $c) ;; the tube is full of a single color
    end
  end
end

bigand $t in [1..$T]: ;; for each tube
  bigand $c in $C: ;; for each color
    bigand $c1 in $C when $c != $c1: ;; for each color
      bigand $e in [1..4]: ;; for each layer
        not p($k, $t, $e, $c) or not p($k, $t, $e, $c1)
        ;; the tube have only one color
      end
    end
  end
end
```

Les résultats sont :

$$\bigwedge_{t\in[1.T]} \bigvee_{c\in C} \bigwedge_{e\in[1..4]} P_{k,t,e,c}$$

$$\bigwedge_{t\in[1..T]} \bigwedge_{c\in C} \bigwedge_{\substack{c1\in C,\\ c\neq c1}} \bigwedge_{e\in[1..4]} \neg P_{k,t,e,c} \vee \neg P_{k,t,e,c1}$$

1.2.2 Règle 2

Pour chaque action *verser* effectuée à une étape i, nous vérifions ses préconditions à l'état $i-1$:

— l'étage e_{src} doit contenir une couleur (il n'est pas vide)

— l'étage e_{dest} doit être vide

— si l'étage e_{dest} est au dessus d'un autre étage, ce dernier doit être de la même couleur que e_{src}

On part du principe que si une action *verser* existe, alors à l'état précédent l'étage source du tube source contient cette même couleur. Ensuite, il faut vérifier que l'étage destination est vide, et qu'il peut donc recevoir une couleur. Enfin, nous vérifions que l'étage en dessous de l'étage destination contient la même couleur que l'étage destination s'apprête à recevoir

```
bigand $e in $ETAPES: ;; for each step
  bigand $t in [1..$T]: ;; for each tube
    bigand $e1 in [1..4]: ;; for each layer
      bigand $t2 in [1..$T] when $t != $t2: ;; for each other tube
        bigand $e2 in [1..4]: ;; for each layer
          bigand $c in $C when $c != vide: ;; for each color
            verser($e, $t, $e1, $t2, $e2, $c)
            => p($e-1, $t, $e1, $c)
            ;; the tube had a color in that layer before pouring
          end
        end
      end
    end
  end
end

bigand $e in $ETAPES: ;; for each step
  bigand $t in [1..$T]: ;; for each tube
    bigand $e1 in [1..4]: ;; for each layer
      bigand $t2 in [1..$T] when $t != $t2: ;; for each other tube
        bigand $e2 in [1..4]: ;; for each layer
          bigand $c in $C when $c != vide: ;; for each color
            verser($e, $t, $e1, $t2, $e2, $c)
            => p($e-1, $t2, $e2, vide)
            ;; the target layer in the target tube is empty
          end
        end
      end
    end
  end
end

bigand $e in $ETAPES: ;; for each step
  bigand $t in [1..$T]: ;; for each tube
    bigand $e1 in [2..4]: ;; for each layer
      bigand $t2 in [1..$T] when $t != $t2: ;; for each other tube
        bigand $e2 in [2..4]: ;; for each layer
          bigand $c in $C: ;; for each color
            verser($e, $t, $e1, $t2, $e2, $c)
            => (p($e-1, $t2, $e2-1, $c))
            ;; the target layer is above a layer of similar color
          end
        end
      end
    end
  end
end
```

Les résultats sont :

$$\bigwedge_{e \in ETAPES} \bigwedge_{t \in [1..T]} \bigwedge_{e1 \in [1..4]} \bigwedge_{\substack{t2 \in [1..T], \\ t \neq t2}} \bigwedge_{e2 \in [1..4]} \bigwedge_{\substack{c \in C, \\ c \neq vide}} verser_{e,t,e1,t2,e2,c} \Rightarrow P_{e-1,t,e1,c}$$

$$\bigwedge_{e \in ETAPES} \bigwedge_{t \in [1..T]} \bigwedge_{e1 \in [1..4]} \bigwedge_{\substack{t2 \in [1..T], \\ t \neq t2}} \bigwedge_{e2 \in [1..4]} \bigwedge_{\substack{c \in C, \\ c \neq vide}} verser_{e,t,e1,t2,e2,c} \Rightarrow P_{e-1,t2,e2,vide}$$

$$\bigwedge_{e \in ETAPES} \bigwedge_{t \in [1..T]} \bigwedge_{e1 \in [2..4]} \bigwedge_{\substack{t2 \in [1..T], \\ t \neq t2}} \bigwedge_{e2 \in [2..4]} \bigwedge_{c \in C} verser_{e,t,e1,t2,e2,c} \Rightarrow P_{e-1,t2,e2,c}$$

1.2.3 Règle 3

Si une action *verser* est réalisée à une étape i, il faut vérifier ses effets dans l'état i :

— l'étage e_{src} est vide ;

— l'étage e_{dest} contient la couleur déplacée

Le principe est similaire à celui de la règles 2 ; si l'action verser existe alors, par conséquent, l'étage source du tube source est vidé, et l'étage destination du tube destination reçoit une couleur.

```
bigand $e in $ETAPES: ;; for each step
  bigand $t in [1..$T]: ;; for each tube
    bigand $e1 in [1..4]: ;; for each layer
      bigand $t2 in [1..$T] when $t != $t2: ;; for each other tube
        bigand $e2 in [1..4]: ;; for each layer
          bigand $c in $C: ;; for each color
            verser($e, $t, $e1, $t2, $e2, $c)
            => (p($e, $t, $e1, vide) and p($e, $t2, $e2, $c))
          end
        end
      end
    end
  end
end
```

Les résultats sont :

$$\bigwedge_{e \in ETAPES} \bigwedge_{t \in [1..T]} \bigwedge_{e1 \in [1..4]} \bigwedge_{\substack{t2 \in [1..T], \\ t \neq t2}} \bigwedge_{e2 \in [1..4]} \bigwedge_{c \in C} verser_{e,t,e1,t2,e2,c} \Rightarrow (P_{e,t,e1,vide} \wedge P_{e,t2,e2,c})$$

1.2.4 Règle 4

Nous vérifions que si un étage vide dans l'état i - 1 reçoit une couleur dans l'état i alors il y a une action *verser* à l'étape i qui déplace cette couleur depuis un étage d'un autre tube. pour résoudre ce problème, nous partons du principe inverse de la règle précédente ; c'est-à-dire qu'au lieu de "générer" l'action verser en réponse aux conditions, nous allons "générer" les conséquences en fonction de l'action verser.

```
;; rules 4
bigand $e in $ETAPES when $e != 0: ;; for each step
  bigand $t in [1..$T]: ;; for each tube
    bigand $e1 in [1..4]: ;; for each layer
      bigand $c in $C when $c != vide: ;; for each color
        (p($e-1, $t, $e1, vide) and p($e, $t, $e1, $c)) =>
        ;; if the tube is empty and after the color is here
          bigor $t2 in [1..$T] when $t != $t2:
          ;; for at least one other tube
            bigor $e2 in [1..4]: ;; for at least one layer
              verser($e, $t2, $e2, $t, $e1, $c)
              ;; then we can move the color
            end
          end
      end
    end
  end
end
```

Les résultats sont :

$$\bigwedge_{\substack{e \in ETAPES, \\ e \neq 0}} \bigwedge_{t \in [1..T]} \bigwedge_{e1 \in [1..4]} \bigwedge_{\substack{c \in C, \\ c \neq vide}} \left((P_{e-1,t,e1,vide} \wedge P_{e,t,e1,c}) \Rightarrow \bigvee_{\substack{t2 \in [1..T], \\ t \neq t2}} \bigvee_{e2 \in [1..4]} verser_{e,t2,e2,t,e1,c} \right)$$

1.2.5 Règle 5

Nous vérifions que si un étage contenant une couleur dans l'état i - 1 devient vide dans l'état i alors il y a une action verser à l'étape i qui déplace cette couleur vers un étage d'un autre tube. Nous réutilisons le principe de la règle 4.

```
bigand $e in $ETAPES when $e != 0: ;; for each step
  bigand $t in [1..$T]: ;; for each tube
    bigand $e1 in [1..4]: ;; for each layer
      bigand $c in $C when $c != vide: ;; for each color
        (p($e-1, $t, $e1, $c) and p($e, $t, $e1, vide)) =>
        ;; if the tube is empty and after the color is here
          bigor $t2 in [1..$T] when $t != $t2:
          ;; for at least one other tube
            bigor $e2 in [1..4]:
            ;; for at least one layer
              verser($e, $t, $e1, $t2, $e2, $c)
              ;; then we can move the color
            end
          end
      end
    end
  end
end
```

Les résultats sont :

$$\bigwedge_{\substack{e\in ETAPES,\\ e\neq 0}} \bigwedge_{t\in[1..T]} \bigwedge_{e1\in[1..4]} \bigwedge_{\substack{c\in C,\\ c\neq vide}} \left((P_{e-1,t,e1,c} \ \wedge \ P_{e,t,e1,vide}) \Rightarrow \bigvee_{\substack{t2\in[1..T],\\ t\neq t2}} \bigvee_{e2\in[1..4]} verser_{e,t,e1,t2,e2,c} \right)$$

1.2.6 Règle 6

Nous vérifions que si un étage contient une couleur dans l'état i - 1 alors dans l'état i, il ne peut qu'être vide ou conserver sa couleur.Une fois la syntaxe de TouIST relativement maîtrisée, modéliser cette règle n'a pas posé de problèmes

```
bigand $e in $ETAPES when $e != 0: ;; for each step
  bigand $t in [1..$T]: ;; for each tube
    bigand $e1 in [1..4]: ;; for each layer
      bigand $c in $C when $c != vide: ;; for each color
        p($e-1, $t, $e1, $c) => ;; if the tube have a color
          (p($e, $t, $e1, vide) or p($e, $t, $e1, $c))
          ;; after each step, any filled layer is either empty
          ;; or has the same color as before
      end
    end
  end
end
```

Les résultats sont :

$$\bigwedge_{\substack{e\in ETAPES,\\ e\neq 0}} \bigwedge_{t\in[1..T]} \bigwedge_{e1\in[1..4]} \bigwedge_{\substack{c\in C,\\ c\neq vide}} P_{e-1,t,e1,c} \Rightarrow (P_{e,t,e1,vide} \vee P_{e,t,e1,c})$$

1.2.7 Règle 7

Pour la dernière règle, nous nous assurons qu'au plus une action *verser* est réalisée à chaque étape. Nous obtenons ainsi un gros bloc de code nécessaire afin de vérifier que l'action ne puisse pas se produire 2 fois dans la même étape.

```
;; rules 7
bigand $e in $ETAPES when $e != 0: ;; for each step
  bigand $t in [1..$T]: ;; for each tube
    bigand $e1 in [1..4]: ;; for each layer
      bigand $c in $C: ;; for each color
        bigand $t2 in [1..$T]: ;; for at least one other tube (target tube)
          bigand $e2 in [1..4]: ;; for at least one other layer (target layer)
            bigand $t3 in [1..$T]:
            ;; for at least one other tube (source tube)
              bigand $e3 in [1..4]:
              ;; for at least one other layer (source layer)
                bigand $t4 in [1..$T]:
                ;; for at least one other tube (source tube)
                  bigand $e4 in [1..4]:
                  ;; for at least one other layer
                    bigand $c2 in $C when $c2 != $c
                    or $t != $t3 or $e1 != $e3
                    or $t2 != $t4 or $e2 != $e4:
                    ;; for each color
                      verser($e, $t, $e1, $t2, $e2, $c)
                      => not verser($e, $t3,
                      $e3, $t4, $e4, $c2) ;; no other pour can be true
                    end
                  end
                end
              end
            end
          end
        end
      end
    end
  end
end
```

Les résultats sont :

$$\bigwedge_{\substack{e\in ETAPES,\\ e\neq 0}} \bigwedge_{t\in[1..T]} \bigwedge_{e1\in[1..4]} \bigwedge_{c\in C} \bigwedge_{t2\in[1..T]} \bigwedge_{e2\in[1..4]} \bigwedge_{t3\in[1..T]} \bigwedge_{e3\in[1..4]} \bigwedge_{t4\in[1..T]} \bigwedge_{e4\in[1..4]} \bigwedge_{\substack{c2\in C\\ c2\neg c\vee t\neg t3\vee e1\neg e3\vee t2\neg t4\vee e2\neg e4}} verser_{e,t,e1,t2,e2,c} \Rightarrow \neg verser_{e,t3,e3,t4,e4,c2}$$

Quand finalement nous avons exécuté le code, nous avons vu que les règles fonctionnaient mais le résultat ne correspondait pas aux attentes. Nous pensions donc que les règles étaient incorrectes, ça nous a donc pris du temps pour comprendre qu'il faillait rajouter les règles de l'exercice 1 pour que cela fonctionne correctement. Un autre problème était qu'au bout de $k = 10$ TouIST retournait un **out of memory** mais quand nous regardions avec un moniteur système nous voyons que TouIST n'utilisait pas l'entièreté de la RAM à sa disposition, c'était donc une contrainte logicielle. Toutefois, le professeur nous a bien confirmé que le problème était solvable avec $k = 10$.

1.3 Exercice 3

Tout d'abord, il a fallu nous adapter au changement de logique, en effet, nous ne travaillons plus avec la "fonction" verser mais avec plusieurs "sous-fonctions" ; *tube_ source*, *tube_ destination* et *couleur_ deplacee*. Nous nous sommes inspirés des contraintes formulées dans l'énoncé ;

1.3.1 Règle 1

Sans doute une des plus complexes à mettre en place - nous ne savons à l'heure actuelle toujours pas si son implémentation est correcte et si elle fonctionne. Nous l'avons divisée en plusieurs parties ; la première (code ci-dessous) s'assure qu'à chaque étape (i), il y ait au moins 1 sous-fonction de chaque type.

```
;; rule 1 (at least 1 of each)
bigand $i in $ETAPES when $i!=$k: ;; for each state
  bigand $t in [1..$T]:
  ;; for each layer
    bigand $t2 in [1..$T] when $t2 !=$t:
    ;; for each layer different from the previous one
      bigand $c in $C when $c != vide:
      ;; for each color different from the previous one
        tube_source($i, $t) and tube_destination($i, $t2) and couleur_deplacee($i,$c)
        ;; if a tube is the source, it can't be the source of another tube
      end
    end
  end
end
```

Les résultats sont :

$$\bigwedge_{\substack{i \in ETAPES \\ i \neq k}} \bigwedge_{t \in [1..T]} \bigwedge_{\substack{t2 \in [1..T] \\ t2 \neq t}} \bigwedge_{\substack{c \in C \\ c \neq vide}} (tube\_source_{i,t} \wedge tube\_destination_{i,t2} \wedge couleur\_deplacee_{i,c})$$

la seconde partie, divisée en 3 sous-parties, s'assure de l'unicité des sous-fonctions par étape.

```
;; rule 1 (1 source max)
bigand $i in $ETAPES: ;; for each state
  bigand $t in [1..$T]: ;; for each layer
    bigand $t2 in [1..$T] when $t2 !=$t:
    ;; for each layer different from the previous one
      tube_source($i, $t) => not tube_source($i,$t2)
      ;; if a tube is the source, it can't be the source of another tube
    end
  end
end

;; rule 1 (1 destination max)
bigand $i in $ETAPES: ;; for each state
  bigand $t in [1..$T]: ;; for each layer
    bigand $t2 in [1..$T] when $t2 !=$t: ;; for each layer different from the previous one
      tube_destination($i, $t) => not tube_destination($i,$t2)
      ;; if a tube is the destination, it can't be the destination of another tube
    end
  end
end

;; rule 1 (1 couleur d\'eplac\'e max)
bigand $i in $ETAPES: ;; for each state
  bigand $c in $C when $c != vide: ;; for each color
    bigand $c2 in $C when $c2 != vide and $c2 !=$c:
    ;; for each color different from the previous one
      couleur_deplacee($i,$c) => not couleur_deplacee($i,$c2)
      ;; if a color is the color to move, it can't be the color to move of another color
    end
  end
end
```

Les résultats sont :

$$\bigwedge_{i \in ETAPES} \bigwedge_{t \in [1..T]} \bigwedge_{\substack{t2 \in [1..T] \\ t2 \neq t}} (tube\_source_{i,t} \Rightarrow \neg tube\_source_{i,t2})$$

$$\bigwedge_{i \in ETAPES} \bigwedge_{t \in [1..T]} \bigwedge_{\substack{t2 \in [1..T] \\ t2 \neq t}} (tube\_destination_{i,t} \Rightarrow \neg tube\_destination_{i,t2})$$

$$\bigwedge_{i \in ETAPES} \bigwedge_{\substack{c \in C \\ c \neq vide}} \bigwedge_{\substack{c2 \in C \\ c2 \neq vide \wedge c2 \neq c}} (couleur\_deplacee_{i,c} \Rightarrow \neg couleur\_deplacee_{i,c2})$$

1.3.2 Règle 2

Ne nous a pas posé grand problème, il s'agit simplement d'une implication dans le cas d'un tube vide.

```
;; rule 2
bigand $i in $ETAPES when $i != 0: ;; for each state
  bigand $t in [1..$T]:
  ;; for each tube
    bigand $e in [1..4]:
    ;; for each layer
      p($i-1, $t, $e, vide) => not tube_source($i, $t)
      ;; if a layer is empty, it means the whole tube is empty
    end
  end
end
```

Les résultats sont :

$$\bigwedge_{\substack{i \in ETAPES \\ i \neq 0}} \bigwedge_{t \in [1..T]} \bigwedge_{e \in [1..4]} (p_{i-1,t,e,vide} \Rightarrow \neg tube\_source_{i,t})$$

1.3.3 Règle 3

Idem, il s'agit d'une implication concernant un tube rempli cette fois-ci.

```
;;rule 3
bigand $i in $ETAPES when $i != 0: ;; for each state
  bigand $t in [1..$T]:
  ;; for each tube
    bigand $e in [1..4]:
    ;; for each layer
      bigand $c in $C:
      ;; for each color
        p($i-1, $t, $e, $c) => not tube_destination($i, $t)
        ;; if a layer contains a color, it means the whole tube is full
      end
    end
  end
end
```

Les résultats sont :

$$\bigwedge_{\substack{i \in ETAPES \\ i \neq 0}} \bigwedge_{t \in [1..T]} \bigwedge_{e \in [1..4]} \bigwedge_{c \in C} (p_{i-1,t,e,c} \Rightarrow \neg tube\_destination_{i,t})$$

1.3.4 Règle 4

Relativement complexe à mettre en place également, d'abord divisée en 2 parties qui ont été par la suite fusionnées, définit en quelque sorte les conséquences de la sous-fonction *tube_source*.

```
bigand $i in $ETAPES when $i != 0:
    bigand $t in [1..$T] :
        bigor $e in [1..4] :
            bigand $c in $C when $c != vide :
                tube_source($i, $t) =>((p($i-1, $t, $e+1, vide) xor p($i-1, $t, 4, $c)) and
                    p($i-1, $t, $e, $c)) => couleur_deplacee($i, $c)
            end
        end
    end
end

bigand $i in $ETAPES when $i != 0:
    bigand $t in [1..$T] :
        bigor $e in [1..4] :
            bigand $c in $C when $c != vide :
                ((p($i-1, $t, $e+1, vide) or p($i-1, $t, 4, $c)) and tube_source($i, $t) and
                    p($i-1, $t, $e, $c) and couleur_deplacee($i, $c)) => p($i,$t,$e,vide)
            end
        end
    end
end
```

Les résultats sont :

$$\bigwedge_{\substack{i \in ETAPES \\ i \neq 0}} \bigwedge_{t \in [1..T]} \bigvee_{e \in [1..4]} \bigwedge_{\substack{c \in C \\ c \neq vide}} (tube\_source_{i,t} \Rightarrow ((p_{i-1,t,e+1,vide} \oplus p_{i-1,t,4,c}) \wedge p_{i-1,t,e,c}) \Rightarrow couleur\_deplacee_{i,c})$$

$$\bigwedge_{\substack{i \in ETAPES \\ i \neq 0}} \bigwedge_{t \in [1..T]} \bigvee_{e \in [1..4]} \bigwedge_{\substack{c \in C \\ c \neq vide}} (((p_{i-1,t,e+1,vide} \vee p_{i-1,t,4,c}) \wedge tube\_source_{i,t} \wedge p_{i-1,t,e,c} \wedge couleur\_deplacee_{i,c}) \Rightarrow p_{i,t,e,vide})$$

Il a été envisagé de la copier-coller et modifier pour définir plus clairement la sous-fonction *tube_source* (à l'aide d'une double implication avec ses conditions).

1.3.5 Règle 5

Assez peu de difficultées rencontrées dans l'implémentation, l'implication est rendue évidente grâce à l'énoncé, bien que notre premier ait été de coder l'implication dans le mauvais sens.

```
bigand $i in $ETAPES when $i != 0:
    bigand $t in [1..$T] :
        bigor $e in [1..4] :
            bigand $c in $C when $c != vide :
                tube_destination($i, $t) and ((p($i-1, $t, 1, vide) xor p($i-1, $t, $e-1, $c))
                    ) => p($i,$t,$e,$c) and couleur_deplacee($i, $c)
            end
        end
    end
end
```

Les résultats sont :

$$\bigwedge_{\substack{i \in ETAPES \\ i \neq 0}} \bigwedge_{t \in [1..T]} \bigvee_{e \in [1..4]} \bigwedge_{\substack{c \in C \\ c \neq vide}} (tube\_destination_{i,t} \wedge ((p_{i-1,t,1,vide} \oplus p_{i-1,t,e-1,c})) \Rightarrow p_{i,t,e,c} \wedge couleur\_deplacee_{i,c})$$

1.3.6 Règle 6

Un peu plus de difficultés rencontrées qu'avec la condition 5, la notion de "plus bas" à été sujette à une mauvaise interprétation au début, de plus, nous ne sommes pas entièrement certains de l'ordre des implications.

```
bigand $i in $ETAPES when $i != 0:
    bigand $t in [1..$T] :
        bigand $e in [1..4] :
            bigand $c in $C when $c != vide :
                p($i-1, $t, $e, vide) and p($i, $t, $e, $c) and couleur_deplacee($i, $c) =>
                    (p($i-1,$t,$e+1, vide) or p($i-1, $t, $e-1, $c)) and tube_destination($i,
                    $t) ;; a layer empty become empty and have have the color move so before
                    it's empty and the other tube is colored
            end
        end
    end
end
```

Les résultats sont :

$$\bigwedge_{\substack{i \in ETAPES \\ i \neq 0}} \bigwedge_{t \in [1..T]} \bigwedge_{e \in [1..4]} \bigwedge_{\substack{c \in C \\ c \neq vide}} p_{i-1,t,e,vide} \wedge p_{i,t,e,c} \wedge couleur\_deplacee_{i,c} \Rightarrow (p_{i-1,t,e+1,vide} \vee p_{i-1,t,e-1,c}) \\ \wedge tube\_destination_{i,t}$$

1.3.7 Règle 7

Probablement la condition ayant posée le plus de problèmes, en effet il est assez difficile d'exprimer dans TouIST ce qu'est "l'étage le plus haut", nous approfondirons ce point plus tard.

```
bigand $i in $ETAPES when $i != 0:
    bigand $t in [1..$T] :
        bigand $e in [1..4] :
            bigor $c in $C when $c != vide :
                bigor $c2 in $C when $c2 != vide:
                    p($i-1, $t, $e, $c) and p($i, $t, $e, vide) =>
                    p($i, $t, $e+1, vide) or p($i, $t, $e-1,c2) ;; if a layer colored become
                        empty so you can have the stage above is empty to or is it can be
                        another color
                end
            end
        end
    end
end
```

Les résultats sont :

$$\bigwedge_{\substack{i \in ETAPES \\ i \neq 0}} \bigwedge_{t \in [1..T]} \bigwedge_{e \in [1..4]} \bigvee_{\substack{c \in C \\ c \neq vide}} \bigvee_{\substack{c2 \in C \\ c2 \neq vide}}$$

$${}_{i-1,t,e,c} \wedge p_{i,t,e,vide} \Rightarrow p_{i,t,e+1,vide} \vee p_{i,t,e-1,c2}$$

1.3.8 Règle 8

Implémentation relativement peu complexe, très similaire à la sous-condition 6 de l'exercice 2.

```
bigand $i in $ETAPES when $i != 0:
    bigand $t in [1..$T] :
        bigand $e in [1..4] :
            bigand $c in $C when $c != vide :
                p($i-1, $t, $e, $c) => p($i, $t, $e, $c) xor p($i, $t, $e, vide) ;; a layer
                    colored can be the same color or empty
            end
        end
    end
end
```

Les résultats sont :

$$\bigwedge_{\substack{i \in ETAPES \\ i \neq 0}} \bigwedge_{t \in [1..T]} \bigwedge_{e \in [1..4]} \bigwedge_{\substack{c \in C \\ c \neq vide}}$$

$$p_{i-1,t,e,c} \Rightarrow p_{i,t,e,c} \oplus p_{i,t,e,vide}$$

1.3.9 Les difficultés

En plus de ces règles, nous avons essayé d'en ajouter d'autres afin de surmonter certaines difficultées rencontrées.

1. La règle des limites
   Durant nos tests, à plusieurs reprises, à l'aide de l'outil de visualisation graphique que nous avons conçu, nous avons pu observer une anomalie ; un étage supplémentaire utilisé par TouIST, aussi après quelque temps passé à essayer de comprendre d'où venait ce problème, nous avons créé cette règle afin d'y remédier. Cependant, après implémentation et tests, il s'est avéré qu'elle causait une erreur, et, de plus, que les étages "fantômes" 0 et 5 ont été créé par TouIST automatiquement, ainsi, la règle, en appelant ces étage, les créent à nouveau, ainsi, après avoir résolu le problème dans le reste du programme (notamment en limitant $\$e$ à [1..3] lorsque $\$e + 1$ était présent ensuite, à [2..4] avec $\$e - 1$), nous avons commenté la fonction.

2. Les règles de l'état précédent
   Durant nos tests, également à plusieurs reprises, nous avons pu constater que TouIST pouvait résoudre le jeu en 1 étape, donc que plusieurs actions avaient lieu dans une même étape, d'où l'idée de créer ces deux règles. Ces règles servent à "renforcer" la règle 1 et s'assurer que les "sous-fonctions" n'entrent pas en jeu en dehors de leur cadre, elles fonctionnent de manière similaire :si à l'état $\$i$-1, un étage d'un tube est vide alors : si c'est l'étage "le plus bas" et qu' il reste vide à l'état $\$i$,alors ce n'est pas un étage tube_destination, cependant, s'il recoit une couleur et qu'il est "le plus bas" de son tube, alors il est l'étage du tube_destination. pour l'état $\$i$. Par contre, si à l'état $\$i$-1 un étage d'un tube est plein, alors : si c'est l'étage "le plus haut" et qu' il reste plein à l'état $\$i$,alors ce

n'est pas un étage du tube_source, cependant, s'il est vidé et qu'il est "le plus haut" de son tube, alors c'est l'étage du tube_destination pour l'état $\$i$.

3. La "fonction" "plus haut"
   A plusieurs reprises lors du codage, nous avons constaté qu'extraire "l'étage le plus haut" (cad l'étage rempli le plus haut dans un tube) était long et peu pratique, mais cependant nécessaire à plusieurs reprises. Ainsi, l'idée de créer un booléen permettant de vérifier si un $\$e$ donné était "l'étage le plus haut" nous est venu. Cependant, mettre cette idée à exécution s'est avérée une tâche fastidieuse est infructueuse, le résultat, divisé en deux sous-partie, n'étant pas fonctionnel.

Informations complémentaires

Comme dit plus tôt, nous nous somme aidés des programmes python suivants pour afficher les tubes des problèmes :

```python
import re
from getch import getch

with open('output.txt') as f:
    datas = f.read()

tubes = re.findall(r"1 p\((\d+),(\d+),(\d+),(\w+)\)", datas)
actions = re.findall(r"1 verser\((\d+),(\d+),(\d+),(\d+),(\d+),(\w+)\)", datas)

colors_b = {
    'vert':  '\033[48;2;087;255;087m',
    'orange': '\033[48;2;255;171;087m',
    'bleu':  '\033[48;2;087;087;255m',
    'rouge': '\033[48;2;255;087;087m',
    'rose':  '\033[48;2;255;087;255m',
    'vide':  '\033[48;2;026;026;026m',
}

colors_f = {
    'vert':  '\033[38;2;087;255;087m',
    'orange': '\033[38;2;255;171;087m',
    'bleu':  '\033[38;2;087;087;255m',
    'rouge': '\033[38;2;255;087;087m',
    'rose':  '\033[38;2;255;087;255m',
    'vide':  '\033[38;2;026;026;026m',
}
#===============#
#===| TUBES |===#
#===============#
# parse to int
for i in range(len(tubes)):
    tubes[i] = (int(tubes[i][0]), int(tubes[i][1]), int(tubes[i][2]), tubes[i][3])

# define size of 3D matrice
size_tubes = [0, 0, 0]
for i in range(len(tubes)):
    if tubes[i][0] > size_tubes[0]:
        size_tubes[0] = tubes[i][0]
    if tubes[i][1] > size_tubes[1]:
        size_tubes[1] = tubes[i][1]
    if tubes[i][2] > size_tubes[2]:
        size_tubes[2] = tubes[i][2]

size_tubes = [size_tubes[0]+1, size_tubes[1], size_tubes[2]]

# create 3D matrice
result_tubes = [[['vide']*size_tubes[2] for i in range(size_tubes[1])] for k in
    range(size_tubes[0])]

# allocate each value in list of tubes in 3D matrice
for e in tubes:
    result_tubes[e[0]][e[1]-1][e[2]-1] = e[3]

tubes = result_tubes
```

```
#=================#
#===| ACTIONS |===#
#=================#
# parse to int
actions_temp = [(None,None,None,None,None)]*(len(actions)+1)
for i in range(len(actions)):
    actions_temp[int(actions[i][0])] = (int(actions[i][0]), int(actions[i][1])-1,
        int(actions[i][2])-1, int(actions[i][3])-1, int(actions[i][4])-1, actions[i][5])
actions = actions_temp

#==============#
#===| SHOW |===#
#==============#
for i in range(size_tubes[0]):
    s = tubes[i]
    space = ' ' if len(str(i)) % 2 == 0 else ''
    sl    = 1 if len(str(i)) % 2 == 0 else 0
    print()
    print("\033[1;31m", end="")
    print(" " * ((6*size_tubes[1] - len(str(i)) - 13 - sl)//2) + "" + ""       + "   "*(10 +
        sl   + len(str(i))) + ""        + "   ")
    print(" " * ((6*size_tubes[1] - len(str(i)) - 13 - sl)//2) + "" + ""         + "   "*(8
        + sl   + len(str(i))) + ""        + "      ")
    print(" " * ((6*size_tubes[1] - len(str(i)) - 13 - sl)//2) + "" + "\033[1;34m" + " TAPE "
        + space + str(i) + " " + "\033[1;31m" + "")
    print(" " * ((6*size_tubes[1] - len(str(i)) - 13 - sl)//2) + "" + ""         + "   "*(8
        + sl   + len(str(i))) + ""        + "      ")
    print(" " * ((6*size_tubes[1] - len(str(i)) - 13 - sl)//2) + "" + ""       + "   "*(10 +
        sl   + len(str(i))) + ""        + "   ")
    print("\033[0m")
    for j in range(size_tubes[1]):
        print("\033[1;31  m  "+colors_b['vide']+" "+"\033[0m\033[1;31  m  \033[0m ", end="")
    print()
    for k in range(size_tubes[2]-1, -1, -1):
        for _ in range(3):
            print("\033[1;31 m \033[1;0m", end="")
            for j in range(size_tubes[1]):
                t = s[j]
                e = t[k]
                char = "   "
                if _ == 0 and ((k+1 < size_tubes[2] and t[k+1] == 'vide') or (k+1 ==
                    size_tubes[2])) and e != 'vide':
                    char = colors_f['vide']+"      " #
                if _ == 1 and k == actions[i][2] and j == actions[i][1]:
                    char = "        "
                    char = colors_f[actions[i][5]]+char
                if _ == 1 and k == actions[i][4] and j == actions[i][3]:
                    char = colors_f['vide']+"      "
                if _ == 2 and k != 0 and e != "vide":
                    char = "        "
                print(colors_b[e] + char + '\033[0m' + ('\033[1;31m\033[1;0m ' if j <
                    size_tubes[1] else '') + ('\033[1;31m\033[1;0m' if j < size_tubes[1]-1 else
                    ''), end="")
            print()
    print(" ", end="")
    for j in range(size_tubes[1]):
        print("\033[1;31   m   \033[1;0m  ", end="")
print()

getch()
```