De la logique pour trouver des trésors

Benjamin Monmege

Un peu de contexte…

- Les bugs… c’est pas cool !
- Pour les éviter, on peut faire des tests…
- Mais parfois, ça ne suffit pas à éviter les bugs, par exemple lorsque le bug se produit rarement ou dépend d’une certaine séquence d’actions d’un utilisateur

Exemple : bug du Zune

- Lecteur MP3 de Microsoft dont la vente a commencé en novembre 2006

- Les heureux détenteurs d'un Zune, qui ont voulu allumer leur appareil le 31 décembre 2008, ont eu la désagréable surprise de rester bloqué sur l'écran d’accueil

- Bug détecté par l’équipe du Zune après quelques heures…

Exemple : bug du Zune

```python
def is_leap_year(year : int) -> bool:
    return (year%4==0 and year%100!=0) or (year%400==0)

def find_date(days : int):
    year = 1980
    while days > 365:
        if is_leap_year(year):
            if days > 366:
                days -= 366
                year += 1
        else:
            days -= 365
            year += 1
    return [days, year]
```

```
>> find_date(10592)
[365, 2008]
```

30 décembre 2008

```
>> find_date(10593)
```

exécution qui ne s’arrête pas !

```
>> find_date(10594)
[1, 2009]
```

1er janvier 2009

Trouver des bugs de manière débranchée

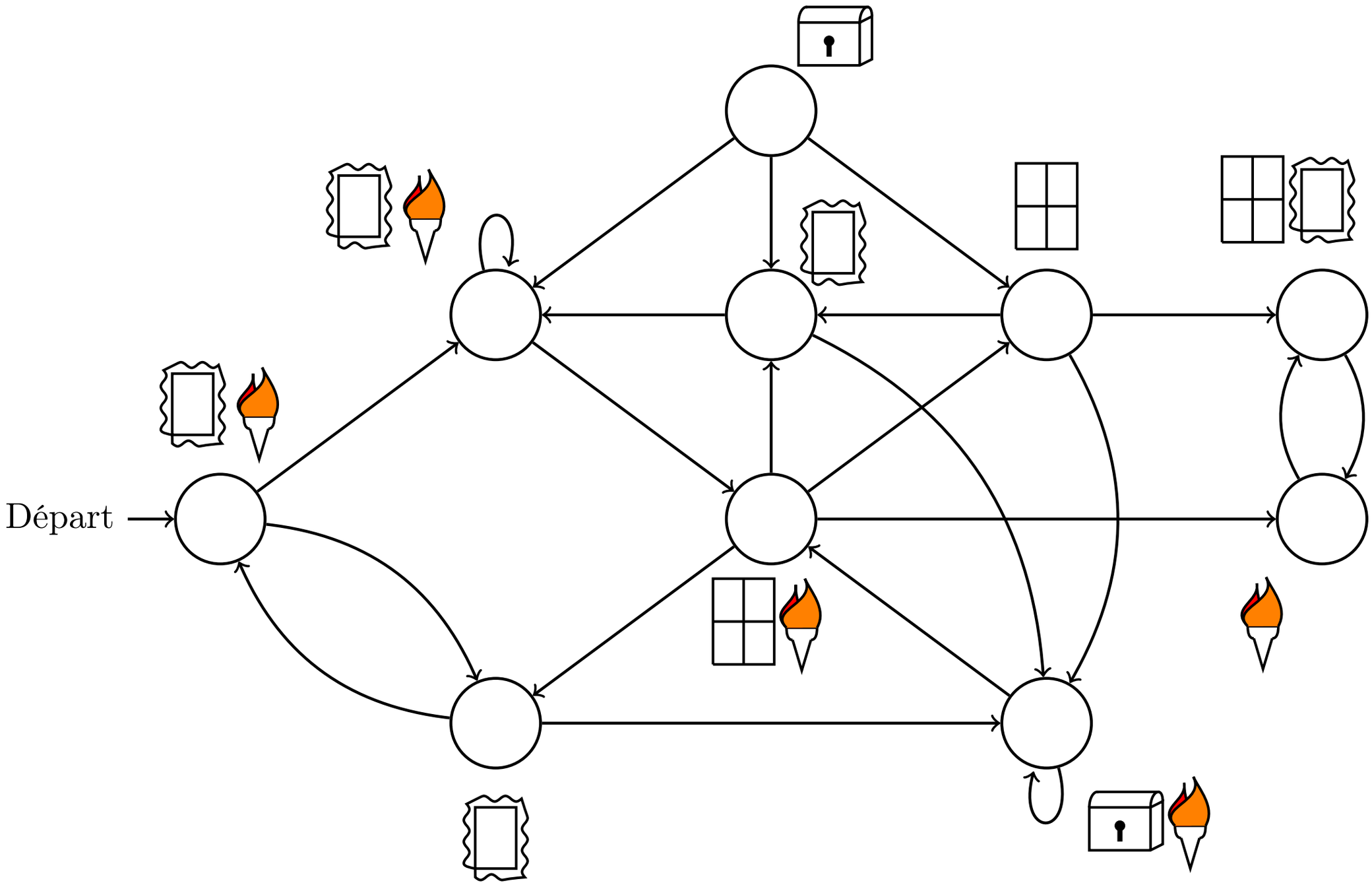
- Découverte d'un langage logique qui permet de spécifier des propriétés, par exemple « toute exécution termine », « à tout moment de toute exécution, on peut toujours redémarrer la machine », « on ne visite jamais la ligne 666 du code »…

- Plutôt que de travailler sur le programme, on va jouer avec un « modèle », ici un graphe fini

- Objectif : découvrir la logique qu'on utilise pour décrire exactement la spécification voulue, pour pouvoir ensuite la vérifier algorithmiquement

activité imaginée et produite par Marie Duflot-Krémer
https://members.loria.fr/MDuflot/

Chasse au trésor dans un château



Comment décrire les formules magiques… euh logiques !

- Des opérateurs qui disent ce qui est vrai dans une salle : torche, trésor, fenêtre, tableau. Un symbole barré pour signifier l’absence de l’objet dans la salle.

- Des opérateurs logiques : ET, OU, NON

Comment décrire les formules magiques… euh logiques !

- Les deux opérateurs comportant une ou plusieurs flèches permettent de parler des chemins qui partent d'une pièce.

  - Celui avec une flèche dit qu'il existe un chemin à partir de cette pièce tel que.... (et il faut écrire la suite de la formule avec d'autres opérateurs).

  - Celui avec plusieurs flèches dit que pour tout chemin qui part de cette pièce on a ... (et pareil la suite reste à écrire)

Comment décrire les formules magiques… euh logiques !

Cette logique parle de chemins infinis ! On ne peut pas choisir de s'arrêter quand ça nous arrange…

- Les quatre derniers opérateurs servent à dire ce qu'il se passe le long d'un chemin : « un jour », « toujours », « dans la prochaine pièce », « jusqu'à ce que ».

- Il y a aussi des parenthèses pour préciser dans quel ordre appliquer les opérateurs.

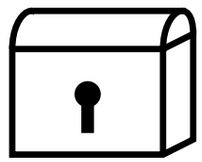
Quelques exemples

À vous d’écrire des formules

- Je peux accéder à un trésor.
- Je peux aller dans une pièce où je pourrai correctement admirer un tableau.
- Toute balade dans le château permet de voir au moins une fenêtre.
- Je peux me balader dans uniquement des pièces avec des torches.
- Je peux arriver devant un tableau en exactement 3 déplacements.
- Il y a moyen de se perdre dans le château dans un endroit à partir duquel je ne peux plus atteindre le trésor.
- Quelqu’un qui a peur du noir peut atteindre le trésor (les gens qui ont peur du noir ne peuvent aller dans les pièces où il n’y a ni torche ni fenêtre).
- Je peux atteindre le trésor en passant d’abord par des salles ayant toutes un tableau, puis des salles ayant toutes une torche, puis le trésor.

Comment vérifier si la formule est vraie dans mon château ?

ça parait dur, vu qu'on parle de chemins infinis et qu'il y en a beaucoup !!

- Propriété : « Je peux accéder à un trésor. » → 

- Pièces qui satisfont la propriété *facilement* ?

Mettons-leur un point dedans…

- N'y a-t-il pas d'autres pièces pour lesquelles on peut dire *facilement* qu'à partir d'elles on peut aller dans les pièces à trésor ?

Mettons-leur aussi un point dedans…

- On continue !

Model-checking

Tant que *j'ai des pièces avec un point dedans* **Faire**
    Choisir une pièce **p** avec un point dedans
    Remplacer le point dans **p** par une croix
    `# pour dire qu'on s'en est occupé`
    Pour *chaque pièce qui a une porte qui arrive dans* **p Faire**
        Si *il n'y a ni point ni croix dans cette pièce* **Alors**
            Y mettre un point
            `# S'il y avait déjà quelque chose on sait que la pièce ne sera pas oubliée`
        Finsi
    Finpour
Fintantque

Algorithme termine

Complexité polynomiale

Algorithmes similaires pour les autres constructions…

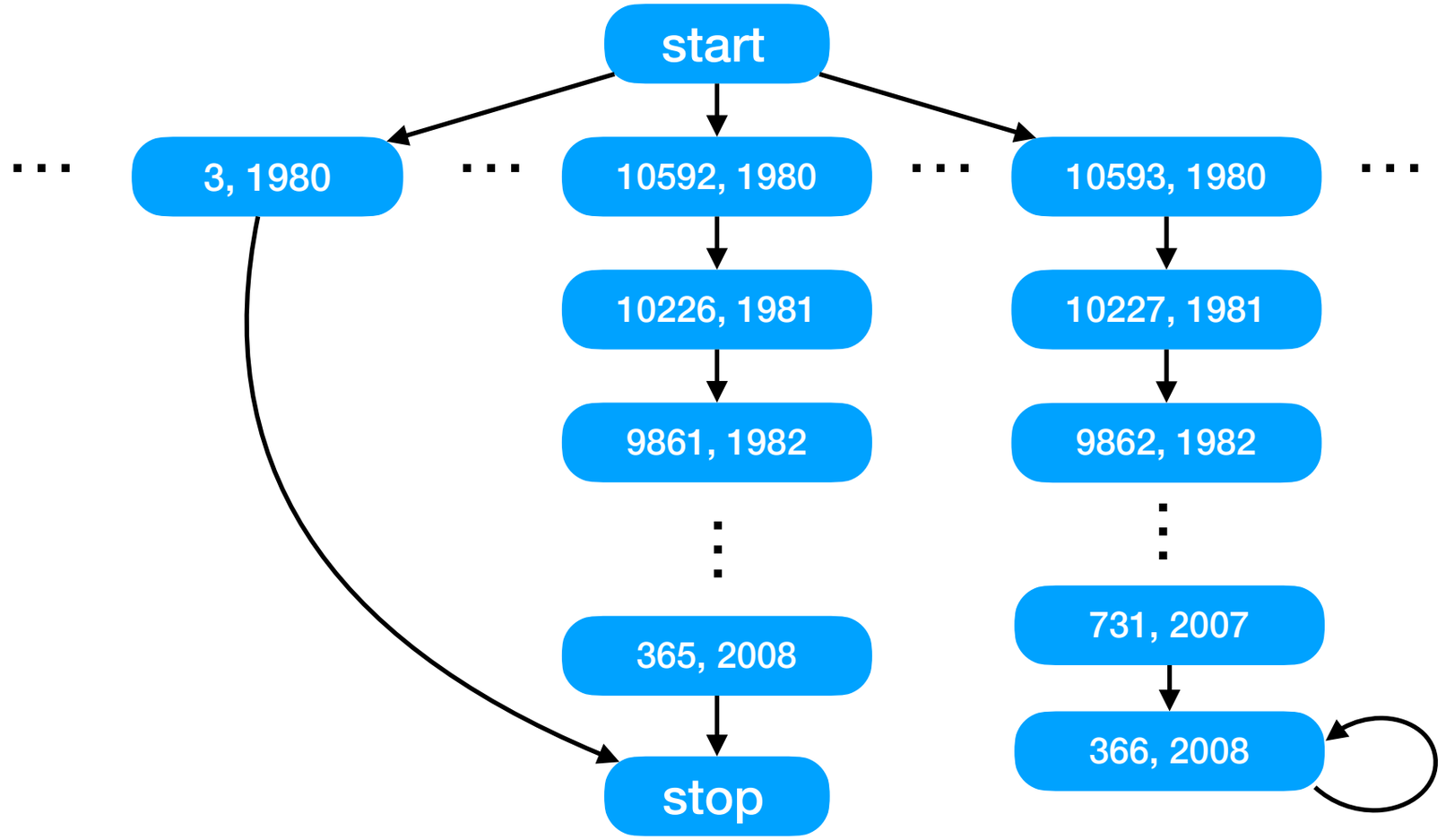
Application au Zune

Programme

```
def is_leap_year(year : int) -> bool:
    return (year%4==0 and year%100!=0) or (year%400==0)

def find_date(days : int):
    year = 1980
    while days > 365:
        if is_leap_year(year):
            if days > 366:
                days -= 366
                year += 1
        else:
            days -= 365
            year += 1
   return [days, year]
```

Graphe des exécutions possibles



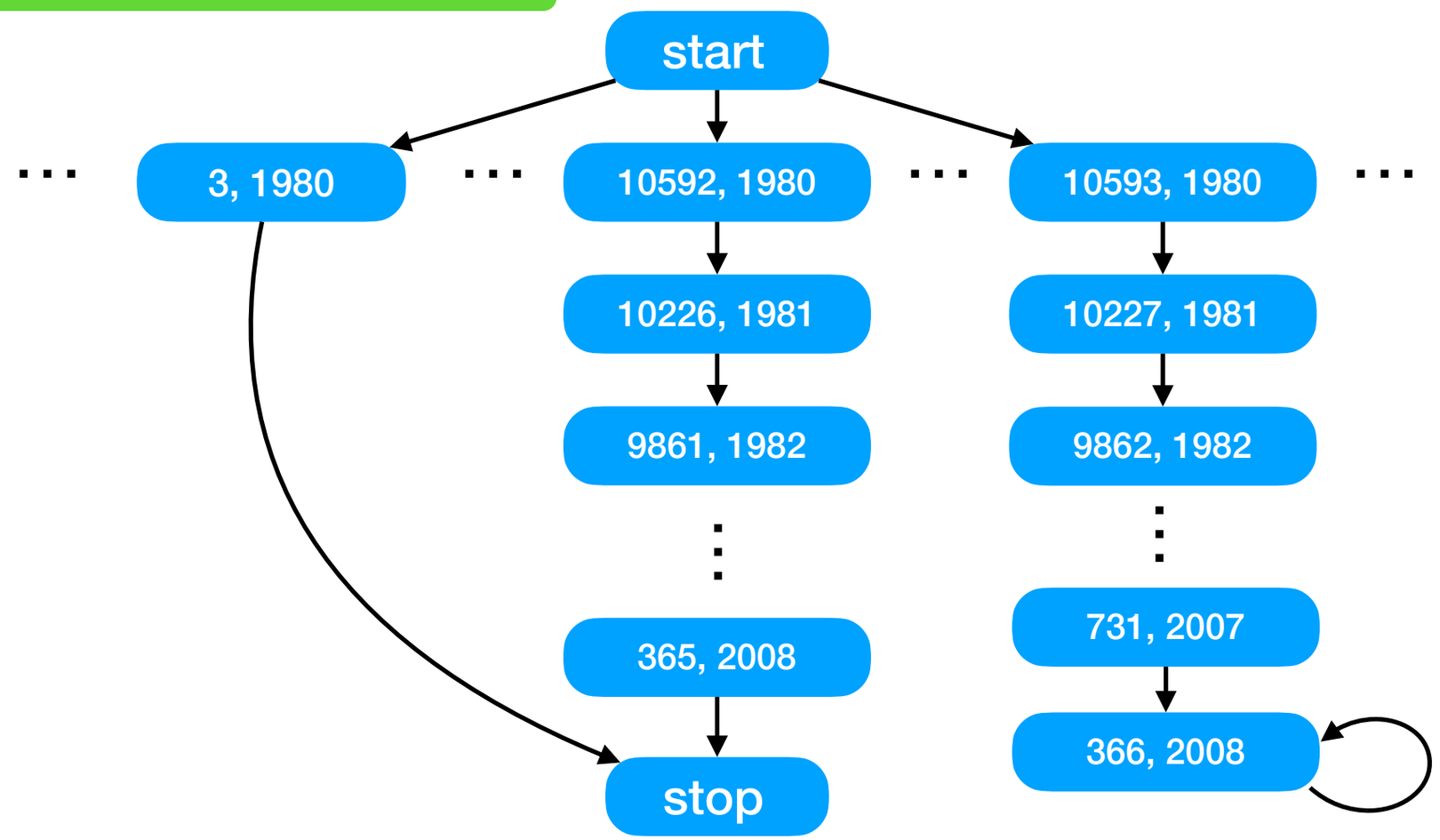
Application au Zune

Spécification $\rightsquigarrow \bullet \; (\; \longrightarrow \; \bullet \; \text{stop} \;)$

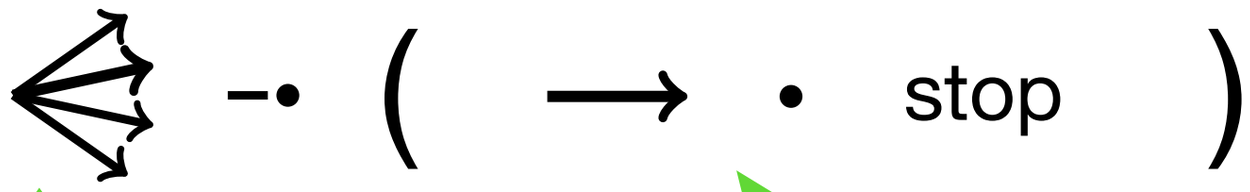
Quelle que soit la première étape (c'est-à-dire la valeur d'appel)

… il y a une exécution qui mène à l'état stop

Graphe des exécutions possibles



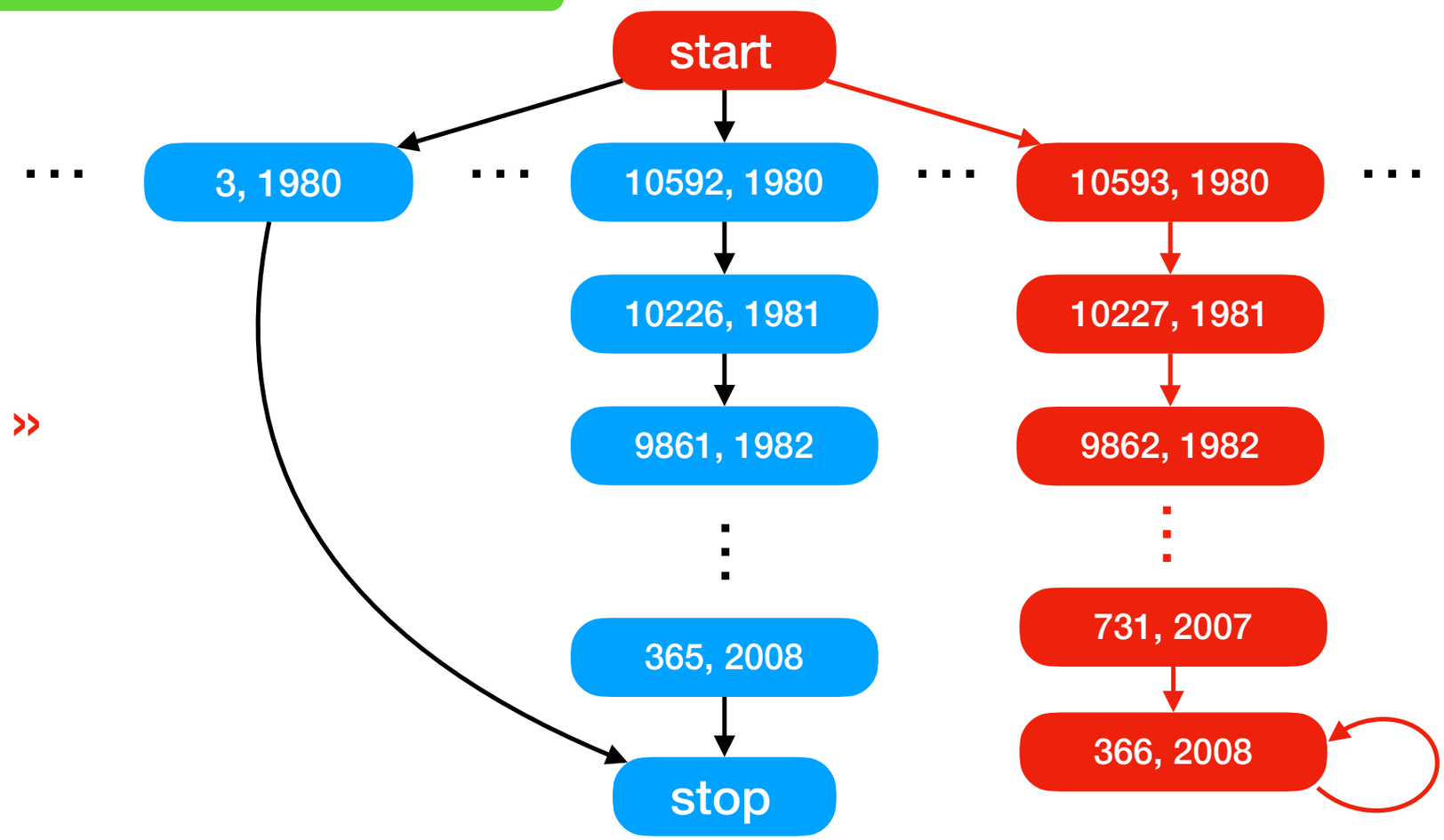
Application au Zune



Quelle que soit la première étape (c’est-à-dire la valeur d’appel)

… il y a une exécution qui mène à l’état stop

L’algorithme de « model checking » renvoie une exécution qui ne satisfait pas la spécification…



Patch

```
def day_count_in_year(year):
    """Renvoie le nombre de jours dans l'année year"""
    if is_leap_year(year):
        return 366
    else:
        return 365


def find_date(days : int):
    """Programme corrigé"""
    year = 1980
    while (days > day_count_in_year(year)):
        days -= day_count_in_year(year)
        year += 1
    return [days, year]
```

Utilisation ?

- Testé (par Marie Duflot-Kremer) auprès d'élèves de troisième lors d'une fête de la science, et de lycéens en première (pas spécialement NSI)

- Testé (par moi) auprès d'enseignant.e.s de NSI

- Utilisation pour aborder ludiquement les rudiments de logique (et, ou, non) avec des formules basiques ?

- Utilisation pour aborder différemment les arguments de terminaison et le calcul de complexité d'algorithme dans des graphes finis (très similaires à des parcours de graphe)