Tony Hoare

Un bref apercue

Par Théo.H

‹There are two methods in software design. One is to make the program so simple,
there are obviously no errors.
The other is to make it so complicated,
there are no obvious errors.›

Édition scientifique avec LaTeX

21 février 2022

Table des matières

Une bref introduction

Tony Hoare ou *Charles Antony Richard Hoare* est un Informaticien influent du XX siecle. Il est principalement connue pour sont implication sur la synchronisation des processus avec les moniteur, La logique de Hoare pour des preuve formelle de justesse algorithmique et le *quicksort*.

1 La logique de Hoare

1.1 Présentation

Le but de la *Logique de Hoare* est de permettre un systeme formelle de raisonement sur la justesse d'un programme Plus concrettement comme chacun le sait il existe des systeme critique pour lequel l'incertitude n'est pas acceptable. Cette logique est baser sur les **triplet de Hoare**, ce triplet de la forme $\{P\}S\{Q\}$ est respectivment composer de :

- ➢ P Une precondition
- ➢ S Le programme
- ➢ Q Une postcondition

Les precondition est postcondition sont deux assertion appartenant a la logique des predicats. Un triplet de hoare est correct si la condition initial P est verifier est S executer ce qui implique que Q est vrai.

exemple :

$$\{x = 5\}x := x \times 2\{x > 0\}$$

Ce triplet est clairement correct, en effet si $x = 5$ est que x est multplier par deux, x est bien sur superieur a 0.

1.2 Méthode

Ce triplet est un outils essentielle car il permet grace a de nombreuse propriete (voir plus bas), de realiser des preuve rigoureuse de nos algorithme. Il y a cependant un problème important et il s'agit de S, en effet dans le cas d'une boucle nous cherchons qu'elle que chose de complexe a trouver un *invariant de boucle* par exemple :

```
int somme(int n):
  int s = 0;
  for(int i=0;i<n+1;i++)
      s++;
  return s;
```

Ici l'invariant de boucle est $s = \sum_{k=0}^{i-1} k$

Maintenant qu'elle que propriete :

Propriété 1.1: Axiome de l'affectation

L'affectation est l'instruction x := E , associant à la variable x la valeur de l'expression E.

$$\overline{\{P[E/x]\}\ x := E\ \{P\}}$$

Propriété 1.2: Règle de composition

La règle de composition s'applique pour les programmes S et T s'ils sont exécutés séquentiellement, où S s'exécute avant T. Le programme issu de cette exécution est noté $S; T$.

$$\frac{\{P\}\ S\ \{Q\}\ ,\ \{Q\}\ T\ \{R\}}{\{P\}\ S; T\ \{R\}}$$

Propriété 1.3: Règle de la conditionnelle

La règle de la conditionnelle permet de combiner deux programmes dans un bloc $'si...fin\ si'$, lorsque les conditions le permettent.

$$\frac{\{P\}\ S\ \{Q\}\ ,\ \{Q\}\ T\ \{R\}}{\{P\}\ S; T\ \{R\}}$$

$$\textbf{si } B \textbf{ alors } S \textbf{ sinon } T \textbf{ fin si } \{Q\}$$

Le reste des propriete peuvent etre retrouver ici

1.3 Application

Maintenant cette logique de Hoarepeut sembler interesante pour un mathematicien est sans doute stimulante pour l'infomaticien avide d'impressioner ces paire mais pourtant des application concrete, essentielle et récente existe.

Avant de parler de l'une de ces aplication concrete une definition :

Definition 1.1 *Un language est dit **Type-Safe** si il ne permet pas les erreur de type.*

```
exemple:
a="Toulon" ;a++

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

errors: you cannot add to an
str, it has no sens...
moron
```

En generale il existe une connexion entre la *Type-Safety* est les problème de mémoire. Bien sur tout cela peut sembler eminament utile pour un programmes ou meme un language, mais certaine chose peuvent sembler plus essentielle que d'autre par exemple un OS.

C'est ce que a essayer de faire des checheur de Microsoft et du M.I.T en creant un OS X86 type-safe.

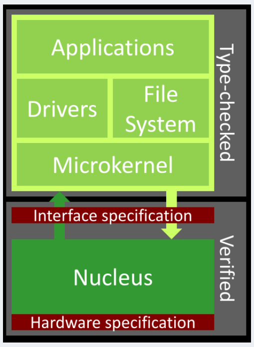


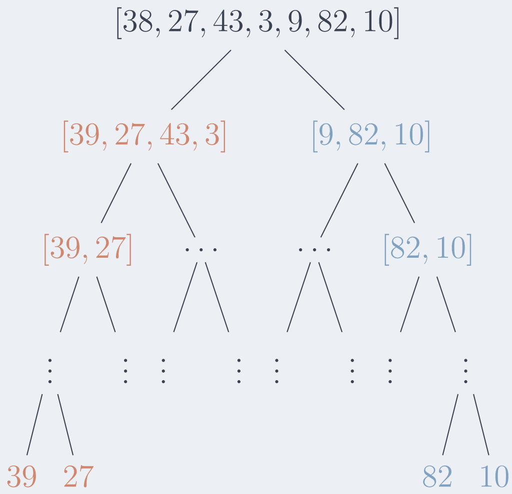
FIGURE 1 – Diagrame de Verve

Cette OS baser bien sur sur la logique de Hoare, a etait cree en utulisante *Boogie* un outils en ligne permettent dans beaucoup de cas, pas touts, de verifier de maniere formelle la justesse d'un algorithme. Il ouvrent bien sur des perspective interesante pour le future.

2 Le Quicksort

2.1 Presentation

Le *Quicksort* ou trie rapide dans la langue de Molière, est un algorithme de trie qui est . . . rapide. De manière Plus concrète il s'agit d'un trie qui se rapproche des limites possible d'un trie comparatifs sans propriete connue.



Ce trie repose sur le principe du diviser pour mieux regner. Si en politique cela consiste a augmenter le nombre de candidats d'oposition, en informatique cela consiste en trois etapes :

1. **Diviser**, on divise l'instance en plus petite instance, en generale en separant en deux ce qui explique le retour recurent du $\log_2$ dans la complexiter de cette famille d'algorithme.
2. **Regner**,cette partie est surement la plus evidente car c'est la que l'ont le trie.
3. **Reunir**, Enfin reunie afin d'obtenir notre resultat final, c'est d'ailleur cette partie qui donne le charactere linaire des trie *"divide and conquere"*

Le trie dans l'exemple n'est pas le trie rapide, mais le trie fusion un trie baser lui aussi sur le *Divide and conquere*, il est a noter que la figure ne reprensente que la phase de division et pas la fusion elle meme, en effet celle ci est specifique au merge sort.

2.2 Principe et implementation

Le trie rapide,comme dit precedament, repose sur le *divide and conquere* plus particulierment autour d'un **pivot**. Ce pivot possede neanmoins une propriete interesante, a la suite d'un *partitionment* tout les element a gauche lui sont inferieur et respectivment, ce que l'ont peut resumer par :

$$\forall (i, j) \in [p, q] \times [q + 1, r], \quad L[i] \leq L[j].$$

Cette propriete implique une infomation importante, **tout les partition sont trier**, ainsi et de manière naturelle. on peut juste ce contenter de reproduire cette procedure de maniere recursive.

```
int Partionner(liste t, uint p, uint r){
  int x=t.t[p];
  int i=p;
  int j=r;
  while (t.t[j]>x)
  {
    j--;
  }
  while (i<j)
  {
    swap(t.t,i,j);
    do
    {
      j--;
    } while (t.t[j]>x);
    do
    {
      i++;
    } while (t.t[i]<x);
  }
  return j;
}

void TriRapide(liste L, uint p, uint r){
  if (p<r)
  {
    q=Partionner(L,p,r);
    TriRapide(L,p,q);
    TriRapide(L,q+1,r);
  }
}
```

2.3 Complexité

De manière étonnante pour un trie utiliser aussi fréquemment sont pire cas est en $\Theta(n^2)$, neanmoins il existe de nombreuse variante de ce trie qui résolve ce problème. Un fix rapide est par exemple le choix du pivot de maniere aléatoire

Concernant ce complexiter dans le meilleur cas l'algorithme `partionner` et en $\Theta(n)$, en effet les deux indice i et j sont respectivement incrementer et decrementer, et ceux jusqu'au pivot donc c'est operation se feront donc

n fois. On peut s'etttoner de ces resultat mitiger et se demander pourquoi cette algo est si utiliser.