13 - Calculabilité

I) Dualité programme données

Dans le modèle de l'architecture de von Neumann est un ordinateur utilise une même structure de stockage pour conserver à la fois les instructions et les données demandées ou produites par le calcul. Dans l'absolu rien n'empêche un programme de produire comme résultat un code d'un autre programme puis de l'exécuter. On parle de **dualité** entre programme et données ce qui signifie que les données peuvent être des programmes et que les programmes sont un type de données comme un autre.

a) Interpréteurs et compilateurs

Un ordinateur ne peut exécuter que du code machine or il est difficile pour un être humain d'écrire du **code machine**. On écrit donc du code source dans un langage évolué tel que le C, le Java, ou encore Python.

Pour que ce code soit exécuté il existe principalement deux méthodes : **l'interprétation et la compilation.**

- Un compilateur est un programme qui prend en entrée un texte dans un langage de "haut niveau" le transforme en code machine (assembleur) qui peut alors être exécuté. Chaque compilation est spécifique à chaque architecture de processeur.
- Un interpréteur est un programme qui prend en entrée un texte dans un langage de "haut niveau", qui analyse qui le transforme en appels à des fonctions de l'interpréteur qui exécutent les instructions du programme.

b) Assembleur MIPS

Le langage assembleur est le langage de plus bas niveau qui représente le langage machine sous forme lisible par un humain. Il est spécifique à chaque type de processeur.

MIPS : Microprocessor without interlocked pipeline stages est une architecture de microprocesseur de type jeu d'instruction réduit RISC

Reduced Instruction-Set Computer. Elle fut développé par la compagnie MIPS Computer Systems Inc. Les processeurs fabriqués selon cette architecture sont surtout utilisés dans les systèmes SGI, dans les systèmes embarqué, comme les ordinateurs de poche, les routeurs Cisco, les consoles de jeux vidéos (N64, PS, PS2 & PSP)

Le jeu d'instruction est basique et relativement simple à manipuler. Pour illustrer les concepts développés ici, nous utiliserons un simulateur multi-plateforme Java nommé Mars.

Trois manière différentes :

- `NomInstruction Opérande_1, Opérande_2, Opérande_3`
- `NomInstruction Opérande_1`
- `NomInstruction`

`NomInstruction` représente le nom symbolique de l'instruction tandis que `Opérande` représente une donnée ou le résultat de l'instruction.

Les opérandes utilisent les modèles d'adressage pour savoir comment obtenir la valeur de la donnée recherchée, ou pour savoir où ranger le résultat de l'instruction.

Exemple :

```
add $t0, $t1, $t2
```

Additionne les valeurs stockés aux adresses mémoires temporaires `t1` et `t2` et stocke le résultat en mémoire tempoire en `t0` . Cette commande va être assemblée pour constituer un nombre binaire de 32 bits.
Télécharger l'archive Java `Mars4_5.jar` le simulateur à l'adresse suivante : https://courses.missouristate.edu/KenVollmar/mars/download.htm

▼ c) Le système d'exploitation

Le système d'exploitation est un programme dont le rôle est de lancer et de contrôler l'exécution d'autre programmes. Si le fichier à exécuter contient du code source, le système d'exploitation détermine quel est l'interpréteur approprié à lancer en lui donnant en entré le fichier du code source à exécuter.

C’est ce qu’il se passe lorsqu’on “double clique” sur `prog.py` , l’interpréteur **python** comme entrée le code source qu’il exécutera.

▼ d) Le téléchargement de logiciel

Lorsqu’un programme est proposer sous forme de code machine il ne pourra pas être exécuté que si le code correspondant au type d’architecture pour laquelle il a été compilé.

Pour contourner ce problème certains langages, comme Java utilisent un code source intermédiaire “byte-code” qui doit être interprété par une “machine virtuelle”

▼ e) Les virus

Les virus sont des programmes conçus pour s’installer et se propager à l’insu de l’utilisateur de l’ordinateur hôte. Les virus utilisent de nombreuses méthodes, dont la plupart consistent d’une manière ou d’une autre à utiliser des données qui sont exécutées comme un programme.

▼ II) Calculabilité

La dualité programme données rend les ordinateurs polyvalents. Ils sont capables d’effectuer un grand types de tâches. On peut donc légitimement se poser la question qu’est-il réellement possible de calculer à l’aide d’un ordinateur ? La capacité de calculer, répondre à des problèmes de manière algorithmique est-elle illimitée ?

▼ a) Fonctions calculables, problèmes décidables

Au début du XXè siècle des mathématiciens ont tenté de formaliser les fondations des mathématiques. L’objectif est de définir une théorie mathématique par un ensemble d’axiomes et de règles formelles puis de vérifier que le modèle est solide.

1. Le modèle n’aboutit pas à des contradictions (cohérence).
2. Toutes propriétés mathématiques vraie peut-être démontrée (complétude).
3. Qu’il existe un algorithme capable de prouver ces propriétés (décidabilité)

Dans les années 30 Kurt Gödel démontre que n’importe quel système logique suffisamment puissant pour décrire l’arithmétique des entiers admet des propositions sur les nombres entiers ne pouvant être ni

infirmées ni confirmées à partir des axiomes de la théorie. Ces propositions sont qualifiés d'indécidables.

En 1936 Alan Turing définit la notion de procédé calculable (algo ou programme) par un modèle théorique appelé `machine de Turing` . On verra que toutes les fonctions ne sont pas calculables, c'est le cas pour le problème dit de l'arrêt. La thèse de Church (mathématicien logicien) affirme que toutes fonctions "effectivement calculable" l'est par une machine de Turing. Thèse de Church peut s'énoncer ainsi "il existe un algorithme répondant à un problème donné si et seulement s'il existe une machine de Turing qui détermine la solution de ce problème."

Tous les langages capables de simuler une machine de Turing ont la même expressivité qu'une machine de Turing. C'est-à-dire les mêmes capacités à calculer. On dit qu'ils sont **Turing-complets**. Pour finir on convient par définition qu'une fonction est **calculable**, ou un problème de décision est décidable, lorsqu'il existe un programme qui renvoie le résultat attendu sur toutes les instances du problèmes.

▼ b) Le problème de l'arrêt

En 1936, Turing montre que le problème de l'arrêt est **indécidable**.

Problème de l'arrêt :

Entrée : un programme P et une entrée E du programme
Renvoie :
`True` si calcul de P sur E se termine, `False` sinon

> 💡 Théorème 1 :
> Le problème de l'arrêt est indécidable.