Interprétation, Compilation, Machines Virtuelles

Pierre Letouzey [1]
letouzey@irif.fr

Irif - Univ. Paris

janvier 2020

1. Merci à Y. Régis-Gianas pour ces transparents

L'art de la programmation

- Informellement, la programmation est :

  L'art de résoudre des problèmes efficacement, par le calcul.

- Il existe de nombreux modèles de calcul que l'on peut par exemple classifier selon les critères suivants :

Critères	*Exemples*
Digital	Ordinateurs standards ou quantiques
Analogique	Réseaux de neurones, mélanges chimiques
Séquentiel	Programme C
Parallèle	Programme Erlang
Automatique	Calcul numérique
Supervisé	Preuve assistée par ordinateur
Fonctionnel	λ-calcul
Impératif	Programme C
Logique	Programme Prolog
Concurrent	Système biologique

Une machine abstraite pour chaque modèle de calcul

- Une machine abstraite décrit l'environnement d'évaluation d'un modèle de calcul.
- Très peu de modèles possèdent une réalisation physique de leur machine abstraite (à l'exception du modèle de Von Neumann).
- On peut cependant émuler une machine abstraite à l'aide d'un autre programme.
- Un tel programme est appelé machine virtuelle.

Les machines virtuelles : une solution miracle ?

- Une machine sans existence physique : ni silicium, ni engrenage.
- Dématérialiser la machine a de nombreuses conséquences.

1. **Raisonnement sur les programmes** : on peut s'abstraire des détails de l'électronique et se donner un cadre formel et universel (indépendant du matériel) pour prouver des propriétés sur l'évaluation des programmes.
2. **Contrôle de l'exécution de la machine** : le programme d'émulation de la machine peut observer un programme avant de l'évaluer pour vérifier qu'il se comporte correctement. Une machine virtuelle peut aussi rattrapper les erreurs, adapter un programme a son environnement, etc.
3. **Manipulation de la machine elle-même comme une donnée** : on peut faire transiter une machine virtuelle à travers internet, la mettre à jour, etc.

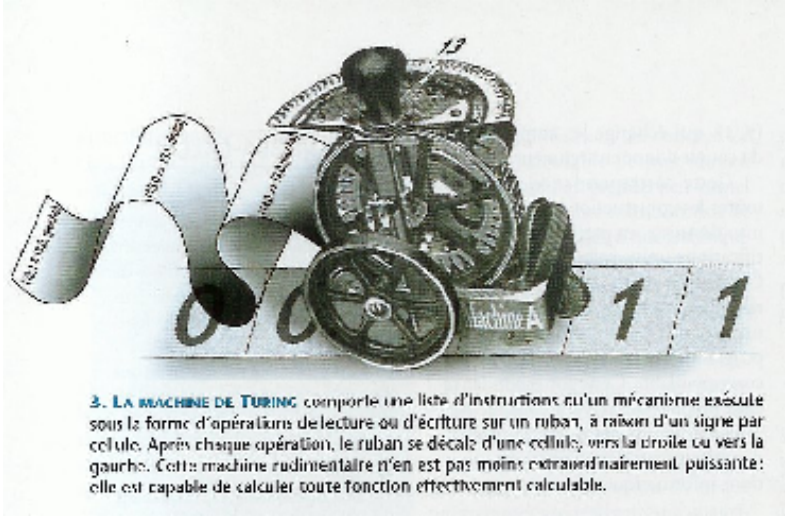
Les machines virtuelles : le revers de la médaille

- L'implémentation de machines virtuelles **efficaces** et **sûres** est difficile.
- Le problème d'efficacité provient de la couche d'interprétation introduite par la machine virtuelle.
- Le problème de sûreté est intrinsèque à la difficulté d'analyser des programmes avant de les évaluer (analyse dite statique).

$\Rightarrow$ L'objectif de ce cours est l'étude des solutions modernes à ces problèmes en termes de logiciel et de matériel.

Quelques rappels sur les machines de Turing

La première de toute : la machine de Turing



3. LA MACHINE DE TURING comporte une liste d'instructions qu'un mécanisme exécute sous la forme d'opérations de lecture ou d'écriture sur un ruban, à raison d'un signe par cellule. Après chaque opération, le ruban se décale d'une cellule, vers la droite ou vers la gauche. Cette machine rudimentaire n'en est pas moins extraordinairement puissante : elle est capable de calculer toute fonction effectivement calculable.

Ingrédients :

- Une bande (infinie) de cases contenant des symboles.
- Une tête de lecture en face d'une case
- Un état courant (p.ex. un numéro d'état)
- Un programme (fini), qui à tout état et symbole lu, associe un nouvel état, un symbole à écrire, et un mouvement (G/D) de la tête de lecture.

Machine de Turing

1. Écrire une addition de bâtonnets : la bande devra passer de « XX+XXXX » à « XXXXXX »
2. Écrire une soustraction de bâtonnets.
3. (dur) Écrire une multiplication de bâtonnets.
4. Donner un exemple de machine ne s'arrêtant pas. Comment savoir à l'avance si on est dans ce cas ?

Machine de Turing et problème de l'arrêt

- Création en 1936 (1er ordinateur : 1945 !).
- Sa motivation, l'Entscheidungsproblem :

  Existe-t-il une machine répondant à toutes les questions oui/non des maths ?
- Réponse négative, via l'indécidabilité du problème de l'arrêt : pas de méthode générale pour savoir si une machine et une bande initiale vont mener à un arrêt.
- Ici, "méthode générale" = algorithme = machine de Turing
- Ces égalités forment la Thèse de Church-Turing : tout ce qui se calcule (meme en Java 1.5 sur Amd 64 3500+) peut se faire *via* une machine de Turing.

Machine de Turing, Variantes et Ordinateurs

- Il existe de nombreuses variantes de la machine de Turing : alphabet des symboles, bandes multiples, bande bornée d'un côté. La plupart sont équivalentes.
- Remarque sur la bande infinie : c'est juste une convenance.
  Un programme ne peut utiliser une infinité de bande en un temps fini ! Ici, infinie = aussi large que nécessaire.
- Analogie avec un ordinateur :
  - bande = mémoire
  - état = pointeur de code (pc)
  - programme = programme (ou plutôt câblage du CPU)

Machine de Turing universelle

- Une seule machine suffit pour tout calculer !
- Astuce : on cache un codage d'une M.d.T. dans les données de la machine universelle, qui ensuite simule cette M.d.T.
- Autrement dit : on peut écrire en M.d.T un interprète de M.d.T. . .
- Dix ans avant le premier ordinateur polyvalent (encore que, cf Jacquard, Babbage, Pascal. . .).

$\Rightarrow$ Plus de détails ? cours de M1 Calculabilité et Complexité . . .

Implémentation des machines virtuelles

Méthodes d'implémentation

Spécification S d'une machine abstraite
$+$
Programme P écrit pour S $\Rightarrow$ Résultat R
$+$
Entrée I de P

Le problème de l'évaluation : Comment calculer R ?

- L'interprétation : on réalise S par un programme *Interprete*, alors :

$$R = \mathit{Interprete}(P, I)$$

(Mais qui évalue le programme *Interprete* ?)

- La compilation : on traduit le programme P en un programme P' équivalent que l'on sait déjà évaluer :

$$R = P'(I)$$

Comment interpréter efficacement ?

- Trois grandes directions :
  1. Des techniques d'écriture d'interprète.
  2. Le choix de langages simples à interpréter.
  3. La virtualisation.

Comment écrire un interprète ?

- Comment écrire en Scala un interprète d'un petit language ?
- Trois ingrédients :
  - un type Expr mimant les constructions syntaxiques du langage ;
  - un type Result des résultats possibles (int/bool/fun/...), dits aussi *valeurs* ;
  - un environnement Env associant à chaque variable sa valeur.

  (cf. le code par la suite)
- Bilan : **simple**, **portable** mais **très lent**.

Comparaison avec le code « natif »

- Le code « natif » est le langage interprété par la machine hôte.
- C'est le moyen le plus efficace de calculer sur cette machine.
- Regardons de plus près le code binaire à l'aide d'`objdump`.
- Bilan : **non portable** mais **très rapide**.

Un code binaire portable : le code-octet ou bytecode

- Les machines virtuelles des langages OCaml, Java ou .Net sont des approximations raisonnables des machines physiques existantes.
- Leur proximité avec les architectures matérielles sur lesquelles elles sont exécutées permet de réduire au minimum la couche d'interprétation (*i.e.* le travail de l'interprète).

$\Rightarrow$ Nous étudierons précisément l'implémentation de la VM de Java et d'OCaml.

- On peut même totalement supprimer la couche d'interprétation à l'aide de techniques de compilation appliquées juste avant l'évaluation.

$\Rightarrow$ Nous parlerons de ces techniques utilisées dans la JVM et dans des émulateurs.

- Bilan : **relative simplicité**, **portabilité** et **rapidité raisonnable**.

Comment interpréter en toute sécurité ?

Le code-octet, une donnée comme une autre ?

- Le code-octet est une application directe de l'architecture de Von Neumann : le programme peut être vu comme une donnée quelconque.
- Cependant, en pratique, si on télécharge une mise-à-jour sur sa carte bancaire, qui nous garantit qu'elle ne mettra pas à mal l'intégrité de notre compte en banque ?

$\Rightarrow$ En tant que donnée exécutable, le code-octet doit être manipulé avec précaution.

- Les solutions connues :
  - l'authentification de l'origine du code-octet (certificats, etc.) ;
  - l'analyse statique : on rejette les programmes dont on n'a pas pu prouver automatiquement le bon comportement.
  - la *sandbox* (bac à sable) : on virtualise la machine hôte en contrôlant les accès aux informations et périphériques critiques.

Bibliographie

- Cours atypique, donc pas de livre “tout-en-un”.
- Sur Java :
  The Java Virtual Machine
  J. Meyer et T. Downing
- Généralités :
  Virtual Machines
  Ian D. Craig, Springer
- ...