TD2: Calculabilité

Pierre-Alain Fouque

Octobre 2020

Le but de ce TD est de montrer l'équivalence entre le modèle RAM et le modèle de calcul des machines de Turing.

1 Le modèle RAM

Le modèle RAM est un modèle de machine ayant les caractéristiques suivantes. Elle possède:

- une bande d'entrée, divisée en cases, munie d'un dispositif qui permet de lire cette bande d'entrée de gauche à droite case après case. On parle de tête de lecture se déplaçant vers la droite. Cette tête de lecture lit le contenu d'une case à la fois, la case qui se trouve sous la tête de lecture.

- une bande de sortie, laquelle est également divisée en cases, sur laquelle la machine est susceptible d'écrire à l'aide d'un dispositif, une tête d'écriture. Cette tête d'écriture permet de remplir une case à la fois et se déplace vers la droite.

On dira d'une tête de lecture ou d'écriture qu'elle pointe sur une case, ou qu'elle se déplace vers la case suivante à droite.

On considère dans un premier temps que les cases contiennent chacune un entier, aussi grand qu'on veut.

Ces deux organes sont les seuls organes d'entrées/sorties de la machine.

D'un point de vue interne, elle possède:

- des éléments de mémoires numérotées, appelés *registres*, en nombre arbitrairement grand, chacun d'eux étant susceptible de contenir un entier. Les registres sont appelées $r_0, r_1, \ldots, r_n, \ldots$, l'indice étant le numéro du registre, appelé *adresse* de la mémoire. Le registre de numéro zéro jouera un rôle spécial, car c'est dans ce registre que s'effectuera toute l'arithmétique. On appelle ce registre l'*accumulateur*.

- un deuxième ensemble de mémoires numérotées contient de manière inaltérable le programme de la machine, programme qui se présente sous la forme d'une suite d'instructions élémentaires (décrites ci-dessous), chaque instructions se trouvant dans une mémoire numérotée. Par-là même, chaque instruction se trouve affectée d'un numéro qu'on appelle l'étiquette de l'instruction.

- une mémoire qui contient un numéro d'instruction. Cette mémoire, appelée compteur ordinal, est initialisée à 1.

Dans ce type de machine, on accède à chaque case mémoire de registre directement par son adresse. C'est la raison pour laquelle ce type est dénommé "Random Access Memory" (RAM en abrégé).

Le programme de la RAM est constitué d'une suite d'instructions prises parmi les opérations élémentaires suivantes:

- des instructions d'affectations:
  CHARGER opérande
  RANGER opérande

- des instructions arithmétiques:
  INCREMENTER opérande
  DECREMENTER opérande
  AJOUTER opérande
  SOUSTRAIRE opérande
  MULTIPLIER opérande
  DIVISIER opérande

- des instructions d'entrée/sortie:
  LIRE opérande
  ECRIRE opérande

- des instructions de rupture de séquence:
  SAUT A étiquette
  SAUT SI POSITIF étiquette
  SAUT SI ZERO étiquette
  ARRET étiquette

Les instructions se présentent en deux parties, une partie *code opération* et d'une partie *adresse*. Si l'instruction est une instruction de rupture de séquence, l'adresse est l'étiquette d'une instruction du programme, sinon c'est un numéro de registre. Les opérandes peuvent être de 3 types:

- soit c'est un entier n. L'opérande est alors le contenu du registre ayant pour numéro cet entier n.

- soit c'est un entier n précédé de "A" (le A signifie Absolu). L'opérande est alors l'entier n lui-même.

- soit c'est un entier n précédé de "I" (le I signifie indirection). L'opérande est alors le contenu du registre ayant pour numéro l'entier contenu dans le registre de numéro n.

Cet ensemble d'instructions élémentaires est typique des ensembles d'instructions que l'on trouve dans les assembleurs. Il manque cependant des instructions de manipulations de caractères et des instructions de type opération logique que nous n'avons pas introduites pour ne pas alourdir cette présentation.

- $\leftarrow$ désigne le symbole d'affectation; à sa gauche, un entier i désigne le registre de numéro i, à sa droite c'est l'entier i lui-même
- le contenu du registre i est désigné par $< i >$
- $<< i >>$ désigne le contenu du registre ayant pour numéro l'entier contenu dans le registre de numéro i
- ACC désigne le registre de numéro 0
- CO désigne le compteur ordinal

Instruction		signification
CHARGER	n	ACC$\leftarrow < n >$
CHARGER A:	n	ACC$\leftarrow$ la valeur n
CHARGER I:	n	ACC$\leftarrow << n >>$
RANGER	n	$n \leftarrow < ACC >$
RANGER I:	n	$< n > \leftarrow < ACC >$
INCREMENTER	n	$n \leftarrow < n > +1$
DECREMENTER	n	$n \leftarrow < n > -1$
AJOUTER	n	$ACC \leftarrow < ACC > +n$
SOUSTRAIRE	n	$ACC \leftarrow < ACC > -n$
MULTIPLIER	n	$ACC \leftarrow < ACC > \times n$
DIVISER	n	$ACC \leftarrow < ACC > /n$
SAUT A	n	$CO \leftarrow n$
SAUT SI POSITIF	n	$CO \leftarrow n$ si $< ACC > \geq 0$
		$CO \leftarrow CO + 1$ sinon
SAUT SI ZERO	n	$CO \leftarrow n$ si $< ACC >= 0$
		$CO \leftarrow CO + 1$ sinon
ARRET		provoque l'arrêt de l'exécution
LIRE	n	$n \leftarrow$ l'entier qui est sous la tête de lecture et se décale vers la droite
ECRIRE	n	la tête d'écriture écrit l'entier n et se décale vers la droite

Exercice 1:

1. Que fait le programme suivant?

```
      LIRE            0
      SAUT SI ZERO    10
      RANGER          1
4:    RANGER          2
      DECREMENTER     1
      CHARGER         1
      SAUT SI ZERO    11
      MULTIPLIER      2
      SAUT A          4
10:   CHARGER A:      1
11:   ECRIRE          2
      ARRET
```

Les différences essentielles entre ce modèle et les machines existantes sont:

1. on a supposé qu'on pouvait utiliser un nombre arbitrairement grand de registres, alors que cette mémoire est bien sûr limitée dans la réalité; de même, on a supposé qu'on pouvait ranger un nombre arbitrairement grand dans chacun des registres ou chacune des cases des bandes d'entrée et de sortie, alors que cela n'est pas le cas dans la réalité.

2. On a supposé que le programme était dans des mémoires inaltérables distinctes des mémoires de type registre. Ce n'est pas le cas dans la réalité.

Plus accessoires sont les différences suivantes:

3. L'ensemble des instructions élémentaires choisi est plus limité que celui qu'on trouve en général dans les assembleurs.

4. Les données d'entrées ne sont pas entrées sous la forme d'entiers lus, mais sous forme de chaîne de caractères, et de même en sortie. On peut convertir l'un dans l'autre.

L'hypothèse 2) va nous conduire à définir le modèle voisin du modèle RAM appelé le modèle RASP (Random Access with Stored Program). La différence essentielle est que les mémoires dans lesquelles sont rangées le programme ne se distinguent en rien des autres mémoires. Pour la commodité de ce qui suit, on considère que chaque instruction occupe deux registres consécutifs: le premier contient le champ opération de l'instruction, et le second le champ adresse de celle-ci.

Le registre 0 est toujours l'accumulateur et le registre 1 nous servira pour faire des sauvegardes temporaires de l'accumulateur.

Les registre 2 à p (où p est un entier impair) contiennent le programme de la RASP et les registres suivants (à partir de $p+1$) sont les registres libres.

La différence essentielle avec RAM est donc que l'on peut, par programme, modifier un registre qui contient ce programme, et donc modifier ce programme lui-même.

2 Simulation d'une machine par une autre

La machine qui simule mime chacune des instructions de la machine simulée, en effectuant généralement plusieurs instructions qui aboutissent au même résultat global. Lorsqu'on a une machine RAM m de programme P qui utilise lors de l'exécution le registre 0 et les registres 1 à p, on dit qu'une machine RAM m' de programme P' simule le fonctionnement de la première si:

- il existe une injection I de $\mathbb{N}$ vers $\mathbb{N}$ qui envoie 0 sur 0
- chaque instruction de P est remplacée par une suite d'instructions de P' telle que, pour toute exécution de P (pour toute donnée), le contenu de chaque registre $I(r)$ de m' est le même après l'exécution de l'instruction correspondante de P.

Exercice 2:

- Montrer qu'on peut simuler une machine RAM utilisant les registres 0 et de 1 à p par une machine RAM Pk qui utilise les registres de k à $k+p-1$.
- Montrer qu'on peut simuler une RAM par une RASP.
- Montrer que dans un programme RASP, on peut se passer de l'indirection. (Montrer en particulier comment simuler l'instruction MULTIPLIER I: n.)

1) À partir d'une RAM de programme P qui utilise lors de l'exécution le registre 0 et les registres 1 à p, on peut construire une RAM de progamme Pk qui la simule l'injection définie par $I(0) = 0$ et $I(r) = k + r$ pour $r > 0$. C'est trivial lorsqu'on ne se sert pas de l'indirection: il n'est à changer que le nom des registres utilisés. Si l'on utilise l'indirection, il faut ajouter la valeur k (grâce à une instruction AJOUTER A: k) au contenu du registre utilisé pour faire l'indirection. Le même énoncé vaut pour les machines de type RASP.

2) Tout ce qui peut être fait avec une machine RAM peut être fait avec une machine RASP. En effet, il suffit si le programme de la RASP est écrit dans les registres 2 à p d'effectuer une translation de p sur tous les numéros de registres utilisés par la RAM correspondante.

3) Dans le modèle RASP on peut se passer de l'indirection. Soit un programme dans le modèle RASP écrit en utilisant les registres de 2 à p et utilisant lui-même les registres $p+1$ à q. On va construire un nouveau programme qui s'écrit en utilisant les registres de 2 à p', en utilisant lui-même les registres $p'+1$ à $q+p'-p$, en remplaçant chaque instruction d'indirection par 6 instructions n'en comportant pas, de telle sorte que le résultat de ces six instructions soit le même sur chaque registre de numéro $p'+i$ que le résultat de l'instruction simulée sur chaque registre de numéro $p+i$. Clairement, $p'-p$ vaut $2 \times (6-1) \times s$, où s est le nombre d'instructions d'indirection du programme. Voici comment l'on s'y prend:

Supposons qu'il s'agisse de simuler l'une des instructions MULTIPLIER I: n du programme initial, occupant par exemple les registres 30 et 31 de la machine de départ. Pour la

simuler, nous disposons de 6 instructions qui occuperont 12 registres consécutifs de numéros déterminés, par exemple les registres 50 à 61 si il y a eu au-dessus 2 instructions d'indirection. La première de ces instructions va consister à sauvegarder le contenu de l'accumulateur dans le registre 1:

50 RANGER
51 1

La seconde consiste en le chargement dans l'accumulateur du contenu du registre $n+p'-p$ (celui qui correspond au registre n dans le programme initial):

52 CHARGER
53 $n+p'-p$

La troisième augmente de $p'-p$ le contenu de l'accumulateur, qui contient alors exactement l'adresse du registre dans lequel se trouve la valeur à multiplier:

54 AJOUTER A:
55 $p'-p$

La quatrième affecte directement la sixième instruction en modifiant le champ adresse de celle-ci. (C'est ici que l'exécution du programme modifie le programme lui-même):

56 RANGER
57 61

L'avant-dernière restitue le contenu initial de l'accumulateur:

58 CHARGER
59 1

Enfin, on effectue l'opération désirée:

60 MULTIPLIER
61 x

où au départ de l'exécution x est une valeur quelconque, et à chaque exécution de la séquence, si le registre n de la machine de départ contenait l'entier a, x vaut $a+p'-p$ au moment de l'exécution de l'instruction rangée dans les cases 60 et 61.

3 Un modèle RAM rudimentaire

On peut se restreindre à un ensemble d'opérations élémentaires très petit.

Exercice 3:

- Montrer qu'on peut se restreindre aux deux types d'instructions: SAUT SI ZERO et ARRET.
- Montrer qu'on peut se restreindre aux deux types d'instructions arithmétiques: INCREMENTER et DECREMENTER. (Montrer en particulier comment simuler l'instruction MULTIPLIER n). En fait, on peut restreindre plus la RAM et on parlera de RAM rudimentaire dans la section suivante.

1) On peut se restreindre aux deux types d'instructions: SAUT SI ZERO et ARRET. Montrons qu'on peut se passer de l'instruction SAUT A n. Cela peut se faire en utilisant le registre 1 comme registre auxiliaire servant à sauvegarder le contenu de l'accumulateur, en

insérant tout d'abord devant chaque instruction étiquetée la séquence:

```
RANGER    1
CHARGER   1
```

puis en remplaçant l'instruction de saut par:

```
RANGER          1
CHARGER A:      0
SAUT SI ZERO    n'
```

où n' est l'étiquette de l'instruction CHARGER 1 introduite juste devant l'instruction d'étiquette n dans la machine de départ.

Le lecteur est invité à vérifier que l'on peut, de façon analogue, se passer de l'instruction SAUT SI POSITIF.

2) Montrons qu'on peut se passer de l'instruction MULTIPLIER. Exécuter l'instruction MULTIPLIER n revient à ajouter $< n >$ fois le contenu de l'accumulateur. Cela peut se faire en utilisant les registres 1 à 3 comme registres auxiliaires, en remplaçant cette instruction par:

```
          RANGER          1
          CHARGER         n
          RANGER          2
          CHARGER A:      0
          RANGER          3
retour    CHARGER         2
          SAUT SI ZERO    etiq   De la même façon, on peut se passer de l'instruction
          DECREMENTER     2
          CHARGER         3
          AJOUTER         1
          RANGER          3
          SAUT A          retour
etiq      CHARGER         3
```

AJOUTER n (exécuter cette instruction revient à incrémenter $< n >$ fois le contenu de l'accumulateur), etc. On peut donc se restreindre à un modèle de calcul, que nous appellerons modèle RAM rudimentaire.

4 Toute fonction T-calculable est R-calculable

On dit qu'une fonction f est T-calculable (respectivement R-calculable) s'il existe une machine de Turing (respectivement une machine RAM) calculant la fonction f. On suppose que la machine de Turing a une bande infinie dans un seul sens et un alphabet à deux lettres 0 et 1. Les contenus des cases de la machine de Turing, numérotées $0, 1, \ldots$ sont contenus respectivement dans les registres $r2, r3, \ldots$ de la RAM et le numéro du registre correspondant à la case sur laquelle se situe la tête de lecture-écriture se trouve dans le registre $r1$ (qui est initialisé à 2, la tête de lecture étant positionnée au départ sur la case numérotée 0).

Une machine de Turing est dite *normalisée* si elle possède un sous-ensemble A d'états qui vérifie:

- aucun état de A ne figure dans un membre de gauche de régle;
- pour tout état $q \in Q \setminus A$, et toute lettre $x \in \Sigma$, il existe une règle de P ayant q, x pour membre gauche.

Autrement dit, une machine de Turing est normalisée si pour un ensemble d'états A, un calcul s'arrête si et seulement si elle est dans un état de A. Un état de A est alors appelé un état d'arrêt. On peut toujours se limiter à un seul état d'arrêt.

1. Montrer qu'à tout état qi de la machine de Turing normalisée, on associe une partie de programme de la machine RAM qui permet de simuler les actions opérées par la machine de Turing dans cet état selon la lettre lue.

2. Montrer que toute fonction calculable par une RAM rudimentaire peut être calculée par une machine de Turing. On dit qu'une RAM rudimentaire si elle contient les instructions suivantes:

Instruction	Significations
I$< n >$	Incrémenter le contenu du registre n
D$< n >$	décrémenter le contenu du registre n
Z$< n >$	Mettre à zéro le contenu du registre n
E$< n, m >$	Échanger les contenus des registres n et m
T$< n > (i, j)$	Si le contenu du registre n est égal à zéro, aller à l'étiquette i, sinon aller à j
ARRET	Arrêt de l'exécution

4.1 Toute fonction T-calculable est R-calculable

Il s'agit de montrer comment un programme P d'une machine de Turing que l'on supposera à bande infinie dans un seul sens, on déduit le programme d'une machine RAM qui vérifie, pour chaque pas de calcul de la machine de Turing: les contenus des cases de la machine de Turing, numérotées $0, 1, \ldots$ sont contenus respectivement dans les registres r2, r3, ... de la RAM, et le numéro du registre correspondant à la case sur laquelle se situe la tête de lecture-écriture se trouve dans le registre r1 (qui est donc initialisé à 2, la tête de lecture étant positionnée au départ sur la case numérotée 0).

À tout état q_i de la machine de Turing, on associe une partie de programme de la machine RAM qui permet de simuler les actions opérées par la machine de Turing dans cet état, selon la lettre lue.

Par exemple, pour une machine de Turing sur un alphabet à deux lettres 0 et 1, si les règles correspondant à l'état q_i sont:
$(q_i, 1) \to (x, m, q_r)$ et $(q_i, 0) \to (y, m', q_s)$, où les mouvements m et m' peuvent valoir L, R), le morceau de programme correspondant est :

k_i	*CHARGER*	*I:r1*
	SAUT SI ZERO	k'_i
	CHARGER	$A : x$
	RANGER	*I: r1*
	INCREMENTER	*r1 (ou bien selon m DECREMENTER r1)*
	SAUT A	k_r
k'_i	*CHARGER*	*A:y*
	RANGER	*I: r1*
	INCREMENTER	*r1 (ou bien selon m DECREMENTER r1)*
	SAUT A	k_s

Si une règle correspondant à un état q_i qui n'existe pas, c'est-à-dire si un couple (q_i, x) n'est membre d'aucune règle, la machine de Turing s'arrête, et le groupe d'instructions correspondant est alors simplement l'instruction: Arrêt.

Clairement, l'exécution de ce groupe d'instructions simule un pas de calcul de la machine de Turing. Les états de celle-ci correspondent exactement aux étiquettes k_i du programme.

Lorsqu'on a écrit tous ces morceaux de programme (correspondant aux différents états de la machine de Turing), on les met bout à bout, et on peut alors mettre des valeurs numériques à la place des étiquettes. L'exécution du programme entier simule le calcul complet de la machine de Turing.

4.2 Toute fonction R-calculable est T-calculable

Il s'agit de montrer ici qu'étant donné le programme d'une machine RAM, que l'on supposera sans perte de généralité être rudimentaire, on peut construire à partir de celui-ci une machine de Turing qui vérifie, pour chaque pas de calcul de la machine RAM:
si $x0, x1, \ldots$ sont les contenus respectifs des registres $r0, r1, \ldots$, la bande de la machine de Turing contient $\#0 : x0\#1 : x1\# \ldots$ où $\#$ et $:$ sont deux nouveaux symboles.

À chaque instruction du programme de la machine RAM de la forme i: $I < n >$, on associe une machine de Turing t_i normalisée, ayant q_i comme état de départ et q_{i+1} comme état d'arrêt, et qui réalise: t_i explore la bande de gauche à droite et compare le numéro après chaque $\#$ avec n. S'il y a égalité, elle incrémente l'entier qui suit (en décalant au besoin tout ce qui suit vers la droite) et s'arrête.

De la même façon, à chaque instruction du programme de la machine RAM de la forme i: $D < n >$, on associe une machine de Turing normalisée, ayant q_i comme état de départ et q_{i+1} comme état d'arrêt qui réalise: t_i explore la bande de gauche à droite et compare le numéro après $\#$ avec n. S'il y a égalité, elle décrémente l'entier qui suit et s'arrête.

À chaque instruction du programme de la machine RAM de la forme i: $Z < n >$, on associe une machine de Turing t_i normalisée, ayant q_i comme état de départ et q_{i+1} comme état d'arrêt, et qui réalise: t_i explore la bande de gauche à droite et compare le numéro après chaque # avec n. S'il y a égalité, elle remplace l'entier qui suit par la constante 0 et s'arrête.

À chaque instruction du programme de la machine RAM de la forme i: $T < n > (j, k)$, on associe une machine de Turing t_i normalisée, ayant q_i comme état de départ et q_j et q_k comme états d'arrêt, et qui réalise: t_i explore la bande de gauche à droite et compare le numéro après chaque # avec n. S'il y a égalité, elle compare l'entier qui suit avec la constante zéro et s'arrête en q_j s'il y a égalité, en q_k sinon.

On laisse au lecteur le soin de décrire ce qui doit être fait pour une instruction de la forme $i : E < n, m >$, (on peut d'ailleurs toujours se passer de ce type d'instruction).

Il apparaît clairement que chaque pas de calcul de la machine RAM, i.e. exécution d'une instruction de numéro i, va être simulée par le calcul de la machine de Turing t_i correspondante. Si l'on rassemble toutes ces machines de Turing pour ne plus en former qu'une seule, celle-ci simule la machine RAM.