Le codage $< M, w >$ peut aussi n'utiliser qu'un alphabet de deux symboles : il suffit de recoder l'ensemble des symboles utilisés en binaire, ce que nous ne faisons pas pour des raisons de lisibilité.

Théorème 2.8 *Le langage* universel $L_U = \{< M, w > \mid w \in L(M)\}$ *est récursivement énumérable.*

On utilise deux rubans sans perdre de généralité d'après le théorème 2.5.

On utilisera l'alphabet $\Sigma_U = \{0\ 1\ 0'\ 1'\ ;\ ,\ d_\Sigma\ d_\delta\ f_\delta\ \mapsto \rightarrow \downarrow \leftarrow\ B\ \$\}$. On peut sans difficulté ramener cet alphabet à 2 symboles (en plus de B et \$) comme vu en exercice.

La machine universelle commence par écrire la configuration initiale sur le deuxième ruban. Puis, si $\gamma \mapsto_M \gamma'$, alors on peut passer du codage de γ au codage de γ' (sur le 2ème ruban) par M'. Quand M accepte, M' aussi.

Pour la deuxième phase (simulation des mouvements de M), on procède comme suit :

1. Se placer au début de la première transition sur le premier ruban et au début de l'état sur le second ruban.
2. On progresse simultanément sur les deux rubans tant que les symboles sont identiques, tout en marquant les symboles du second ruban.
3. En cas d'échec (mismatch), on progresse jusqu'au début de la transition suivante sur le premier ruban et, sur le deuxième, on revient au début du code de l'état (en démarquant)
4. En cas de succès, on passe à la lecture des symboles, d'une part sur le premier ruban et d'autre part sur le second. En cas d'échec, on procède comme pour les échecs sur les états. En cas de succès (on arrive dans une phase de reconnaissance au symbole $\mapsto$), passer à l'étape suivante
5. Si le résultat de la transition de M est $\rightarrow, a', q'$, on remplace simplement $, q, a$ par a', q', (i.e. on se place juste avant le début de l'état puis on recopie la fin de la transition). Il faut ensuite remplacer éventuellement le symbole blanc qui suit l'état par son code.
6. Si le résultat de la transition de M est $\leftarrow, a', q'$, c'est un peu plus délicat, puisqu'il faut translater le symbole précédant $, q$, de $| < Q > | + 2$ vers la droite ; on utilise ici que les codes des états ont tous même longueur.

Remarquons que les codages sont calculables. Par exemple :

Lemme 2.9 *La fonction qui à* $< M >$ *associe* $< M, < M >>$ *est calculable.*

En effet, il suffit de recoder $< M >$ une deuxième fois en utilisant les codages des symboles 0 et 1.

Théorème 2.10 *L_u n'est pas co-récursivement énumérable (et donc pas récursif).*

$\overline{L_u} = \{< M, w > \mid w \notin L(M)\}$. Si ce langage était r.é, soit M_N une machine de Turing telle que $L(M_N) = \overline{L_u}$. On construit alors une machine de Turing D telle que $L(D) = \{< M > \mid < M > \notin L(M)\}$: sur la donnée $< M >$, D simplement recopie le code de M pour obtenir $< M, < M >>$ puis applique M_N. Mais alors,

$$< D > \in L(D) \Longleftrightarrow < D > \notin L(D)$$

Ce qui est absurde.

2.7 Problème de l'arrêt

Soit $L_{arret} = \{< M, x > \mid M(x) \neq \perp\}$.

Théorème 2.11 *L_{arret} est récursivement énumérable et pas co-récursivement énumérable (donc indécidable).*

Tout d'abord, L_{arret} est récursivement énumérable : considérons la machine M_{arret} qui, étant donné $< M, x >$, simule la machine universelle et, quand celle-ci est sur le point d'accepter ou de rejecter, accepte. M_{arret} accepte L_{arret}.

Si L_{arret} était co-récursivement énumérable, soit $\overline{M_{arret}}$ une machine qui accepte $\overline{L_{arret}}$. On construit alors la machine H qui accepte $\{< M > \mid M(< M >) = \perp\}$ comme suit : H commence par dupliquer $< M >$, écrivant $< M, < M >>$, puis applique $\overline{M_{arret}}$. Si $\overline{M_{arret}}$ est sur le point de s'arrêter en **reject**, H entre dans un état spécial, où H se déplace indéfiniment vers la droite sans rien faire (et donc ne s'arrête pas).

$< H > \in L(H)$ si et seulement si $H(< H >) = \perp$. Mais, comme H ne rejette aucun mot, $H(x) = \perp$ si et seulement si $x \notin L(H)$. Donc $H(< H >) = \perp$ si et seulement si $< H > \notin L(H)$. Absurde. Il n'existe donc aucune machine de Turing qui accepte $\overline{L_{arret}}$.

2.8 Réductions

D'une manière générale, on posera les problèmes de la façon suivante :

Donnée : $D \in S$

Question : Q

où S est un ensemble récursif et $Q \subseteq S$. Il s'agit d'une façon d'écrire l'ensemble $\{D \in S \mid Q\}$ où Q est cette fois vu comme un prédicat. On se passe le plus souvent de parler du codage de la donnée. Par exemple, on écrit

> **Donnée :** M, w où M est une machine de Turing et $w \in \{0,1\}^*$
>
> **Question :** M s'arrête-t-elle sur w ?
>
> est indécidable.

Cet énoncé est une façon de dire que $\{< M, w > \mid M(w) \neq \perp\}$ n'est pas récursif.

Pour montrer qu'un problème, donné sous cette forme, est indécidable, on procède souvent par *réduction* d'un problème connu pour être indécidable : Si P_1 est le problème

Donnée : $D_1 \in S_1$

Question : Q_1

est indécidable, et P_2 est le problème

Donnée : $D_2 \in S_2$

Question : Q_2

Pour prouver que P_2 est indécidable, il suffit d'exhiber une fonction récursive $f : S_1 \to S_2$ telle que, pour toute donnée $D_1 \in S_1$, $D_1 \in Q_1 \iff f(D_1) \in Q_2$. Voici un premier exemple

Proposition 2.12 *Le problème suivant est indécidable :*

L'arrêt universel :

Donnée *: une machine de Turing M*

Question *: M s'arrête-t-elle sur toutes les données ?*

On réduit le problème de l'arrêt : on considère la fonction f qui à $< M, x >$ associe $< M_x >$ où M_x est la machine qui écrit x puis simule M sur x (sans tenir compte de l'entrée). f est une fonction calculable : il suffit en effet d'ajouter$|x| + 2$ états à M, qui permettent d'effacer le ruban et d'écrire x. De plus, M_x s'arrête sur toute entrée si et seulement si M s'arrête sur x.

2.9 Théorème de Rice

Théorème 2.13 *Si $\mathcal{P}$ est une propriété non triviale des langages récursivement énumérables, alors $\mathcal{P}$ est indécidables.*

Nous donnons ci-dessus l'énoncé "classique", mais en voici une version plus précis :

Pour tout ensemble $\mathcal{P}$ de (codes de) machines de Turing tel que :

1. pour toutes machines M, M' si $< M >\in \mathcal{P}$ et $L(M) = L(M')$ alors $< M' >\in \mathcal{P}$
2. on peut exhiber des machines M_1, M_2 telles que $< M_1 >\in \mathcal{P}$ et $< M_2 >\notin \mathcal{P}$

alors le problème suivant est indécidable :

Donnée : $< M >$

Question : $< M >\in \mathcal{P}$?

Pour prouver ce résultat, nous utiliserons une réduction du problème de l'arrêt, un type de réduction qui sera très fréquemment utilisé par la suite.

Soit $\mathcal{P}$ un sous-ensemble non-trivial des langages récursivement énumérables. $\mathcal{P}$ est récursif ssi $\overline{\mathcal{P}}$ est récursif. On peut donc supposer sans perte de généralité que $\emptyset \notin \mathcal{P}$. Comme $\mathcal{P}$ est non vide, soit $L \in \mathcal{P}$ et M_L une machine qui accepte L.

On réduit le problème de l'arrêt

Donnée : $< M, x >$ où M est une machine de Turing sur Σ et $x \in \Sigma^*$

Question : M s'arrête-t-elle sur x ?

au problème de l'appartenance à $\mathcal{P}$ en construisant, à partir de la donnée $< M, x >$ du problème de l'arrêt, une donnée M_x du problème de l'appartenance à $\mathcal{P}$, de telle manière à ce que $M_x \in \mathcal{P}$ ssi M s'arrête sur x.

Soit M_{arret} une machine qui accepte L_{arret}. M_x est une machine qui, sur la donnée y, commence par simuler M_{arret} sur x et, en cas de succès, simule ensuite M_L sur y. On remarque que $L(M_x) \in \{L, \emptyset\}$ et donc $L(M_x) \in \mathcal{P}$ ssi M s'arrête sur x.