Langages formels (théorie élémentaire et applications)

Monoïdes

- **Définition** : Un monoïde, c'est un triplet $\langle M, \circ, 1 \rangle$. M est le support ou domaine du monoïde, c'est-à-dire un ensemble. ° est un opération binaire dans M, associative. 1 est un élément neutre de M.

- **Proposition** : Il existe un et un seul élément neutre.

- **Définition** : Etant donnée deux monoïdes $\langle M, \circ_M, I_M \rangle$ et $\langle N, \circ_N, I_N \rangle$ et *f* une application de M dans N. On dit que *f* est un morphisme lorsque :
  1. $\forall x, y \in M,\ f(x \circ_M y) = f(x) \circ_N f(y)$
  2. $f(I_N) = I_M$

- **Proposition** : Il existe un et un seul élément neutre.

- **Définition** : On dit que $\langle M, \circ_M, I_M \rangle$ est un sous-monoïde de $\langle N, \circ_N, I_N \rangle$ lorsque :
  1. $M \subseteq N$
  2. $I_M = I_N$
  3. $\forall x, y \in M,\ x \circ_M y = x \circ_N y$

- **Proposition :** L'intersection de deux sous-monoïdes est un sous-monoïde.

- **Définition :** Etant donné un monoïde M, étant donné une partie X incluse dans M, on note X^* le plus petit (au sens de l'inclusion) sous-monoïde de M qui contient X.

- **Théorème :** Il existe et c'est l'intersection de tous les sous-monoïdes de M.

- **Théorème** : En notant $X^0 = \{I_M\}$, et $\forall n \in N,\ X^{n+1} = X \circ X^n = \{x \circ y / x \in X, y \in X^n\}$, on a $X^* = \bigcup_{n \geq 0} X^n$.

Le monoïde libre

Soit A un ensemble (fini dans la plupart des cas) appelé alphabet, on note A^* l'ensemble des séquences finies d'éléments de A, on note *v.w* la concaténation de deux telles séquences *v* et *w*.

On dispose alors d'un monoïde $\langle A^*, \cdot, \varepsilon \rangle$. Les éléments de A* sont appelés mots sur l'alphabet A. L'opération est appelé concaténation. ε est l'élément neutre de la concaténation, c'est-à-dire la séquence vide, appelée mot vide.

Dans la suite, on notera $a_1 \ldots a_n$ la séquence $(a_1, \ldots, a_n)$ et pour tout mot $w = a_1 \ldots a_n$, on note $|w| = n$ la longueur du mot *w*.

- **Définition** : On appelle langage sur A toute partie de L inclue dans A*, c'est-à-dire un ensemble de mots finies sur l'alphabet A.

- **Quelques opérations sur les langages** :
  1. $L_1 \bigcup L_2$, noté $L_1 + L_2$
  2. $L_1 \bigcap L_2$
  3. $L_1 \cdot L_2 = \{v \cdot w / v \in L_1, w \in L_2\}$
  4. $L^* = \bigcup_{n \geq 0} L^n$

- **Convention** : On notera abusivement pour toute lettre $a \in A$, a le mot réduit à la seule lettre a mais aussi a le langage réduit au seul mot a. Cette convention nous permet de décrire simplement des langages particuliers appelés langages rationnels.

- **Proposition** : Tout langage défini à l'aide des lettres de l'alphabet A, des opérations +, . , * utilisées un nombre fini de fois est rationnel (ou régulier). Etant donné un alphabet $A = \{a, b\}$, $A^* = (a + b)^*$ est rationnel.

- **Définition :** Etant donné un langage $L \subseteq A^*$ et une lettre $a \in A$, on définit $a^{-1}L$ le résidu de L en a : $\{w \in A^* / aw \in L\}$.

- **Lemme** : …

- **Définition** : On appelle résidu de L tout langage L', tel qu'il existe un mot v sur A avec $L' = \{w \in A^* / vw \in L\}$. On note $L' = v^{-1}L$.

- **Proposition** : …

- **Théorème** : Si L est rationnel alors L admet un nombre fini de résidus.

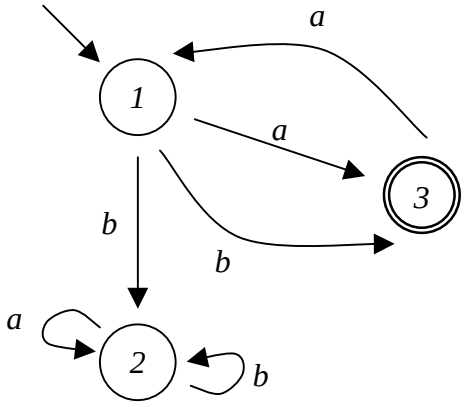
Automates d'états finis

On veut comprendre l'outil informatique qui apparaît à travers la construction des résidus, pour un langage rationnel.

Un automate est un graphe fini orienté dont les arcs sont étiquetés par des lettres et dont on distingue un sommet ou état particulier dit état initial, un ensemble de sommets ou états dits états finaux.

Formellement, un automate est un 5-uplet, $\Lambda = \langle Q, q_0, A, \delta, F \rangle$ avec Q l'ensemble des états, q_0 l'état initial, A un alphabet, δ la fonction de transition, et F l'ensemble des états finaux. La fonction de transition est une application $Q \times A \to \mathrm{P}(Q)$ interprétée par $q_1 \xrightarrow{a} q_2$ si et seulement si on a $q_2 \in \delta(q_1, a)$.

- **Exemple** : Considérons l'automate $\Lambda = \langle \{1,2,3\}, 1, \{a,b\}, \delta, \{3\} \rangle$ avec $\begin{cases} \delta(1,a) = \{3\} \\ \delta(1,b) = \{2\} \\ \delta(2,a) = \{2\} \\ \delta(2,b) = \{2\} \\ \delta(3,a) = \{1\} \\ \delta(3,b) = \{1\} \end{cases}$.



- **Définition** : On définit $\delta^* : Q \times A^* \to \mathrm{P}(Q)$ tel que $\delta^*(q,w) \subseteq Q$ est l'ensemble des états accessibles à partir de l'état *q* en lisant le mot *w*.

  Plus précisément, on définit δ^* par récurrence sur $|w|$:

  1. si $|w| = 0$, c'est-à-dire $w = \varepsilon$, on pose $\delta^*(q,\varepsilon) = \{q\}$
  2. si $|w| = n+1$, c'est-à-dire $w = a.w'$ pour $a \in A$ et $w' \in A^*$ avec $|w'| = n$, on pose $\delta^*(q,a.w') = \bigcup \{\delta^*(q',w') / q' \in \delta(q,a)\}$.

- **Proposition** : On dit qu'un mot *w* est reconnu par l'automate Λ s'il existe une lecture de *w* à partir de l'état initial q_0 jusqu'à un état final.

  Plus précisément, *w* est reconnu par Λ, lorsque $\delta^*(q_0,w) \cap F \neq \mathrm{O}$. On note alors $L(\Lambda) = \{w / \delta^*(q_0,w) \cap F \neq \mathrm{O}\}$ le langage des mots reconnus par Λ.

- **Définition** : On dit qu'un automate Λ est déterministe lorsque pour tout $q \in Q$ et pour tout $a \in A$, $|\delta(q,a)| \leq 1$.

- **Définition** : On dit qu'un automate Λ est complet lorsque pour tout $q \in Q$ et pour tout $a \in A$, $|\delta(q,a)| \geq 1$.

- **Algorithme** : Un automate déterministe reconnaissant un langage L se traduit facilement en algorithme permettant de décider si un mot entré appartient au langage ou pas. Cet algorithme est linéaire par rapport à la taille du mot entré. (…)

- **Proposition** : Si Λ est déterministe mais pas complet, il est facile de construire Λ_c déterministe et complet, tel que $L(\Lambda) = L(\Lambda_c)$. Il suffit de rajouter un état bidon…

- **Théorème de *Kleene - Gui*** : Pour tout langage $L \subseteq A^*$, les trois propositions suivantes sont équivalentes :

  1. L est rationnel.
  2. $L = L(\Lambda)$ pour Λ un automate d'états finis.
  3. $L = L(\Lambda)$ pour Λ déterministe et complet.

- **Méthode pour déterminiser un automate** : Soit $\Lambda = \langle Q, q_0, A, \delta, F \rangle$ avec $\delta : Q \times A \to \mathrm{P}(Q)$. On étend la définition de la fonction de transition en $\overline{\delta} : \mathrm{P}(Q) \times A^* \to \mathrm{P}(Q)$, telle que :
  1. $\forall J \subseteq Q,\ \overline{\delta}(J, \varepsilon) = J$
  2. $\forall J \subseteq Q,\ \forall a \in A,\ \overline{\delta}(J, a) = \bigcup_{q \in J} \delta(q, a)$
  3. $\forall J \subseteq Q,\ \forall w \in A^*,\ \forall a \in A,\ \overline{\delta}(J, w.a) = \overline{\delta}\left(\overline{\delta}(J, w), a\right)$.

  w est reconnu si $\overline{\delta}(\{q_0\}, w) \cap F \neq \mathrm{O}$. En fait, $\overline{\Lambda} = \langle \mathrm{P}(Q), \{q_0\}, A, \overline{\delta}, F' \rangle$ est déterministe. Soit n le nombre d'état de Λ, alors $\overline{\Lambda}$ possède 2^n états. (donner un exemple…)

- **Théorème** : Pour tout automate d'états finis Λ, il existe une expression rationnelle qui définit le langage $L(\Lambda)$.

- **Méthode de définition d'un langage par une expression rationnelle** : Soit L_q les langages reconnus à partir des états q. On établit un système d'équations ensembliste entre les langages, avec $L(\Lambda) = L_{q_0}$. Ce système peut éventuellement avoir plusieurs solutions. La solution qui nous intéresse est la plus petite (au sens de l'inclusion). (donner un exemple de système…)

  La résolution du système se base sur le lemme suivant.

- **Lemme** : Pour tout langage L_1 et L_2, la plus petite solution de $X = L_1.X + L_2$ est $L_1^*.L_2$.

Automates à transitions immédiates, dit à ε - transitions

Un automate à ε - transitions est un 6-uplet, $\Lambda = \langle Q, q_0, A, \delta, F, \mu \rangle$ où $\langle Q, q_0, A, \delta, F \rangle$ est un automate et $\mu : Q \to \mathrm{P}(Q)$ est la fonction des transitions immédiates, symbolisés par $\xrightarrow{\varepsilon}$.

$\forall J \subseteq Q, \overline{\mu}(J)$ représente l'ensemble des états accessibles à partir de J par les ε - transitions.

- **Proposition** : Pour tout langage L, les deux assertions suivantes sont équivalentes.
  1. L est reconnaissable.
  2. L est reconnu par un automate à transitions immédiates normalisé : F est un singleton $\{f\}$, $q_0 \neq f$, $\mu(f) = \mathrm{O}$ et $\forall a \in A, \delta(f, a)$ n'est pas défini.

- **Proposition** : Considérons deux langages reconnaissables, on démontre grâce aux ε - transitions que l'union, la concaténation, l'étoile, l'intersection, le complémentaire, ou la différence sont encore reconnaissables. Par conséquent, tout langage rationnel est reconnaissable.

Propriétés de clôture des langages réguliers

Par définition, la classe des langages rationnels est close par union, produit, et étoile.

- **Lemme** : Le complément d'un langage rationnel est rationnel.

- **Lemme** : L'intersection de deux langages rationnels est rationnel.

Petit retour sur les résidus : automates minimaux

- **Lemme** : Pour tout automate $\Lambda = \langle Q, q_0, A, \delta, F \rangle$, en notant L_q le langage reconnu à partir de l'état q, c'est-à-dire $L_q = \{w \in A^* / \delta^*(q,w) \cap F \neq \mathrm{O}\}$, on a que pour tout mot $w \in A^*$, $w^{-1}L(\Lambda) = \bigcup_{q \in \delta^*(q_0,w)} L_q$.

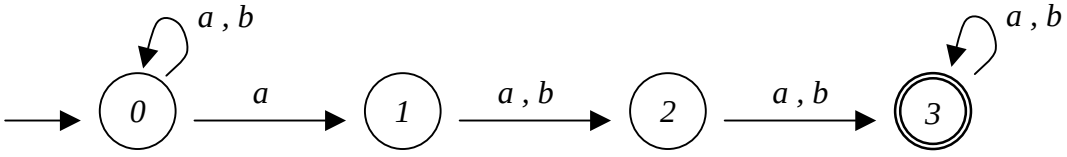
- **Lemme** : Tous langages rationnels admet un nombre fini de résidus.

- **Théorème** : Soit L un langage régulier, il existe un automate Λ déterministe et complet reconnaissant L. On peut effectivement construire L :
  1. construction d'un automate avec ε-transitions
  2. suppression des ε-transitions
  3. déterminisation

- **Corollaire** : Tout langage régulier est reconnaissable par un automate déterministe, complet, ayant un nombre minimum d'états. C'est l'automate des résidus, c'est-à-dire l'automate $\Lambda = \langle Q, q_0, A, \delta, F \rangle$ avec $Q = \{w^{-1}.L / w \in A^*\}$ et $q_0 = \varepsilon^{-1}L = L$ … Dans cet automate, les états sont en fait des langages. (…)

- **Méthode de déterminisation** : Prenons en exemple le langage $L = A^* a A^2$ pour $A = \{a, b\}$. Ce langage est régulier. Par conséquent, il existe un automate déterministe et complet le reconnaissant.

  1. On construit un automate non déterministe (complet) reconnaissant *L*.



  2. On va maintenant déterminiser cet automate, à l'aide d'un tableau. Notons par ailleurs, qu'il possède 2^n états, soit 8 dans le cas présent.

$\longrightarrow$	***a***	***b***
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2, q_3\}$	$\{q_0, q_2, q_3\}$

$\{q_0, q_1, q_2, q_3\}$	$\{q_0, q_1, q_2, q_3\}$	$\{q_0, q_2, q_3\}$
$\{q_0, q_2, q_3\}$	$\{q_0, q_1, q_3\}$	$\{q_0, q_3\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0, q_3\}$
$\{q_0, q_3\}$	$\{q_0, q_1\}$	$\{q_0\}$

- **Méthode de minimisation** : Etant donné un automate déterministe et complet, on cherche à construire un automate minimal reconnaissant le même langage. Une idée consiste à regrouper toute paire d'état à partir desquelles le même sous-langage est reconnu.

- **Algorithme** : (…)

Vers la sûreté de fonctionnement des programmes

La correction d'un programme vérifie la correction partielle d'une part, et la terminaison d'autre part.

- **Correction partielle** : "Si le programme termine, les valeurs produites sont correctes pour toutes entrées valides." On le montre en exhibant *l'invariant* de boucle. Si il ne se termine pas, il est considéré comme partiellement correct !

- **Terminaison** : Pour la terminaison, en toute généralité, on cherchera "une quantité", appelée *variant*, régulièrement consommée par le programme qui s'épuisera nécessairement. La recherche du variant, et plus généralement le problème de la terminaison est un problème compliqué qui ne peut pas être résolu par un ordinateur.

- On dispose par ailleurs d'un outil théorique important, *la logique de Hoare*, qui va nous permettre de formaliser ces considérations.

Syntaxe et sémantique des while-programmes

- **Définition** : Un while-programme est construit à partir :
  - d'un ensemble de variables entières
  - des opérations arithmétiques (+, -, *, /, !, ^, …)
  - des opérations booléennes $(\wedge, \vee, \neg)$
  - des opérations de comparaisons (≤, ≥ , >, =, …)
  - de l'affectation $x := t$ avec t un entier
  - de la forme conditionnelle : *if e then* P_1 *else* P_2, avec *e* une expression conditionnelle, P_1 et P_2 deux while-programmes
  - de la composition : P_1 ; P_2 avec P_1 et P_2 deux while-programmes
  - de la boucle "tant que" : *while e do P*, avec *e* une expression conditionnelle et *P* un while-programme

- **Définition** : A tout while-programme *P*, on associe S_P la sémantique de *P*, qui est une fonction de l'ensemble des états possibles de notre machine (virtuelle sous-jacente) dans ce même ensemble.

  Un état stable est une fonction sur l'ensemble des variables à valeur dans l'ensemble des entiers naturels. L'état instable est symbolisé par $\perp$ (exemple d'une boucle infinie).

  Autrement dit, $S_P : \begin{cases} (Var \to N \cup \{\perp\}) \to (Var \to N \cup \{\perp\}) \\ \sigma \mapsto S_P(\sigma) \end{cases}$ avec $S_P(\perp) = \perp$, où σ représente l'environnement de départ et $S_P(\sigma)$ l'environnement d'arrivée.

- **Sémantiques diverses** :
  - composition : $S_{P_1;P_2}(\sigma) = S_{P_2}(S_{P_1}(\sigma))$
  - affectation : $S_{x:=t}(\sigma) = \sigma([x \to t])$ avec $(\sigma([x \to t]))(y) = \begin{cases} t \; si \; y = x \\ \sigma(y) \; si \; non \end{cases}$
  - (…)

- **Sémantique du *while*** : Considérons le while-programme *P* suivant : *while e do P_1*. On définit $S_P^n(\sigma)$ le résultat de l'exécution de *P* à partir de σ avec au plus *n* passages de boucles. Si le programme ne termine pas en moins de *n* passages de boucles alors le résultat est $\perp$. Ce qui donne $S_P^0(\sigma) = \begin{cases} \sigma \text{ si } \sigma(e) \text{ est faux} \\ \perp \text{ si non} \end{cases}$ et $S_P^{n+1}(\sigma) = \begin{cases} \perp \text{ si } \sigma = \perp \\ \sigma \text{ si } \sigma(e) \text{ est faux}. \\ S_P^n(S_{P_1}(\sigma)) \text{ si non} \end{cases}$

- **Lemme**: Soit σ, soit *k*. Si $S_P^k(\sigma) \neq \perp$, alors $\forall n \geq k, S_P^n(\sigma) = S_P^k(\sigma)$. Autrement dit : si l'on s'arrête au bout de *k* passage, les environnements suivants restent inchangés.

  Sémantique du *while* (suite) : D'après le lemme précédent, on propose la définition suivante : $S_P(\sigma) = \begin{cases} \perp \text{ si } \forall k, S_P^k(\sigma) = \perp \\ S_P^k(\sigma) \text{ si non avec k suffisament grand pour que } S_P^k(\sigma) \neq \perp \end{cases}$. On a $S_P(\sigma) = \perp$ si le programme ne termine pas, sinon on a $S_P(\sigma) \neq \perp$.

Décrire la correction partielle d'un programme : les assertions de Hoares

[On ne s'occupe pas ici de la terminaison du programme.]

Etant donné φ une propriété d'environnement, on note $\sigma \models \varphi$ le fait que l'environnement σ satisfait la propriété φ.

- **Définition** : On appelle assertion de Hoare une expression de la forme $\{\varphi\}P\{\psi\}$ avec *P* un while-programme, φ et ψ des propriétés d'environnement.

- **Définition** : L'assertion $\{\varphi\}P\{\psi\}$ est valide lorsque pour tout environnement σ, si $\sigma \models \varphi$ et $S_P(\sigma) \neq \perp$, alors nécessairement $S_P(\sigma) \models \psi$.

- **Règles de raisonnement sur les assertions** : On formule ces règles de la manière suivante : $\frac{H_1, H_2 \ldots H_n}{C}$, où $H_1, H_2 \ldots H_n$ représente une liste d'hypothèses (logiques ou assertions) et *C* la conclusion.

  - règle pour l'affectation : $\frac{\varphi \Rightarrow \psi[x \to t]}{\{\varphi\}x := t\{\psi\}}$
  - règle pour la composition : $\frac{\{\varphi\}P_1\{I\}, \{I\}P_2\{\psi\}}{\{\varphi\}P_1; P_2\{\psi\}}$ pour *I* une propriété intermédiaire appropriée.
  - règle pour *if* : $\frac{\{\varphi \wedge e\}P_1\{\psi\}, \{\varphi \wedge \neg e\}P_2\{\psi\}}{\{\varphi\}\text{if } e \text{ then } P_1 \text{ else } P_2\{\psi\}}$
  - règle du *while* : $\frac{\varphi \Rightarrow I, \{I \wedge e\}P\{I\}, (I \wedge \neg e) \Rightarrow \varphi}{\{\varphi\}\text{while e do } P\{\psi\}}$ pour *I* un invariant appropriée.

- **Exemples de recherche d'un invariant** : (…)

- **Théorème (cohérence & complétude)** : Toute assertion prouvable est valide (cohérence), et réciproquement toute assertion valide est prouvable (complétude).

- **Théorème** : Le calcul de Hoare est cohérent.

- **Complétude sémantique** : Toute assertion de Hoare valide admet un arbre de preuve dont toutes les hypothèses sont des affirmations logiques vraies.

- **Complétude logique** : Toute affirmation logique vraie est prouvable (dans un système approprié).

- **Exemple de preuve** : Considérons le while-programme *P* défini par : *if* $x < y$ *then* P_1 *else nop*, avec P_1 : $\begin{bmatrix} z \coloneqq x \\ x \coloneqq y \\ y \coloneqq z \end{bmatrix} : P_2$. On cherche à prouver la validité de l'assertion $\{true\}P\{y \le x\}$. Pour ce faire, on construit un arbre de preuve : on procède en remontant assertion par assertion, en utilisant les règles de raisonnement déjà exposées.

  1. On commence par utiliser la règle du *if* : $\dfrac{\{x < y\}P_1\{y \le x\},\ \overbrace{\{\neg(x < y)\}nop\{y \le x\}}^{évident}}{\{true\}P\{y \le x\}}$. La seconde hypothèse étant immédiate. Il reste à prouver que la première hypothèse est valide.
  2. Prouvons donc $\{x < y\}P_2; y \coloneqq z\{y \le x\}$. La règle de composition s'exprime de la manière suivante : $\dfrac{\{x < y\}P_2\{I\},\{I\}y \coloneqq z\{y \le x\}}{\{x < y\}P_2; y \coloneqq z\{y \le x\}}$, avec *I* une propriété intermédiaire adaptée, non encore définie à ce moment du raisonnement.
  3. Penchons-nous pour commencer sur la seconde hypothèse $\{I\}y \coloneqq z\{y \le x\}$, quant à la première nous y reviendrons. Il suffit ici d'appliquer la règle d'affectation, c'est-à-dire $\dfrac{I \Rightarrow (y \le x)[y \to z]}{\{I\}y \coloneqq z\{y \le x\}}$. $I \Rightarrow (y \le x)[y \to z]$ est banalement vrai en prenant $I : z \le$. A partir de ce point, *I* est déterminé.
  4. Il reste maintenant à prouver $\{x < y\}P_2\{I\}$, avec $I : z \le x$ …

- **Cas de la récursivité** : On définit le while-programme d'une fonction $f(x : y)\,corps$ avec *corps* un while-programme, avec *x* l'ensemble des paramètres d'entrées, et *y* la valeur retournée. On cherchera donc à prouver la formule suivante $\{\varphi\}f(x : y)\,corps\{\psi\}$. De manière générale, on suppose que chaque appel récursif à *f* dans le corps se déroule correctement… La règle pour la récursivité s'exprime : $\dfrac{(\{\varphi\}f(x : y)\{\psi\}) \Rightarrow (\{\varphi\}\,corps\{\psi\})}{\{\varphi\}f(x : y)\,corps\{\psi\}}$.

- **Exemple de la factorielle** : …

Méthode des préconditions les plus faibles pour prouver les assertions de Hoare

Supposons que $\{\varphi\}P\{\psi\}$ est valide, comment la prouver ? (différence entre validité et vérité d'une assertion ???)

- **Définition** : Etant donné un while-programme *P* et ψ une propriété d'environnement, on note $wp(P,\psi)$ (weaked precondition) la propriété d'environnement la plus faible telle que, pour tout environnement σ vérifiant $wp(P,\psi)$, si $S_P(\sigma) \neq \perp$, alors on a $S_P(\sigma) \models \psi$.

  En fait, comme $wp(P,\psi)$ est la propriété la plus faible satisfaisant cela, on a pour tout environnement σ, $\sigma \models wp(P,\psi)$ si et seulement si $S_P(\sigma) \neq \perp \Rightarrow S_P(\sigma) \models \psi$.

- **Théorème** : Si on dispose d'un langage L de description de propriétés d'environnement qui est suffisamment riche pour que, pour tout programme *P*, toute propriété $\psi \in L$, $wp(P,\psi) \in L$ (vérifier ?), alors on peut toujours ramener l'étude d'une assertion de Hoare à l'étude d'un nombre fini de propriété d'environnement (le calcul de Hoare est sémantiquement complet).

Un mot sur la détection de la terminaison des programmes

Pour tout programme *P*, on note $\lceil P \rceil$ le codage du texte définissant *P*. Existe-t-il un programme *Q* détectant la terminaison de tout programme *P* passé en paramètre ? Plus précisément, existe-t-il *Q* tel que, pour tout environnement σ :

1. $S_Q(\sigma) \neq \perp$, c'est-à-dire *Q* termine ;
2. si $\sigma(x) = \lceil P \rceil$ pour un programme *P* quelconque, alors $S_Q(\sigma)(y) = \begin{cases} 1 \; si \; S_P(\sigma) \neq \perp \\ -1 \; si \; S_P(\sigma) = \perp \end{cases}$;
3. si $\sigma(x) \neq \lceil P \rceil$ pour tout programme *P*, alors $S_Q(\sigma)(y) = 0$.

- **Théorème :** Il n'existe pas de tel programme *Q*. Ce théorème montre que le problème de l'arrêt d'un programme est indécidable. En revanche, il est semi-décidable. C'est-à-dire qu'il existe *Q* tel que, pour tout environnement σ, pour tout programme *P* qui se termine, si $\sigma(x) = \lceil P \rceil$ alors $S_Q(\sigma)(y) = 1$.

- **Méthode** : Il n'y a pas de méthode systématique ! L'idée directrice consiste à exhiber une suite décroissante. On cherche à montrer que cette suite est finie, ce que l'on exprimera par une formule de Hoare, avant que de la démontrer.

  Etudions la terminaison du petit programme suivant : *while i > c do P*. Le variant est une exprimé en fonction des variables du programme. Ici, c'est bien naturellement *i*. La formule de Hoare à prouver sera donc : $\{i = j \wedge i \geq c\} P \{i < j \wedge i \geq c\}$. Dans des cas plus généraux, la formule de Hoare nécessite des enrichissement : des préconditions comme les conditions d'entrée de boucle du *while*, et des postconditions…

- **Exemple de la recherche dichotomique** : (…)

- **Définition** : $(E,<)$ est bien-fondé si toute suite strictement décroissante est finie. Par exemple, $(N,<)$ est bien-fondé ; $(Z,<)$ n'est pas bien-fondé ; $(X^*,<)$ avec < l'ordre lexicographique n'est pas bien-fondé.