une instruction fictive `giveup ;` telle que la fonction Σ ne soit jamais définie en `(giveup ;,e,m)`, on peut définir la suite d'approximations finies de l'instruction `while (b) q`.

p_0 `= if (b) giveup ; else skip ;`

p_1 `= if (b) {q if (b) giveup ; else skip ;} else skip ;`

...

p_{n+1} `= if (b) {q` p_n`} else skip ;`.

L'instruction p_n tente d'exécuter l'instruction `while (b) q` en faisant au maximum `n` tours de boucle. Si, après `n` tours, elle n'a pas terminé, alors elle abandonne.

Il n'est pas difficile de démontrer que pour tout entier `n` et état `e, m`, si Σ`(`p_n`,e,m)` est définie, alors pour tout `n'` supérieur à `n`, Σ`(`$p_{n'}$`,e,m)` est définie également et Σ`(`$p_{n'}$`,e,m) =` Σ`(`p_n`,e,m)`. Cela traduit le fait que si l'instruction `while (b) q` termine quand le nombre de tours autorisés est `n`, elle termine également, et dans le même état, quand le nombre de tours autorisés est `n'`.

Il y a donc deux possibilités pour la suite Σ`(`p_n`,e,m)` : ou bien elle n'est jamais définie ou bien elle est définie à partir d'un certain rang, et dans ce cas, elle est constante sur son domaine. Dans ce second cas, on appelle *limite* de la suite, la valeur qu'elle prend sur son domaine. La suite n'a en revanche pas de limite si elle n'est jamais définie. On peut maintenant définir la fonction Σ dans le cas où l'instruction `p` est de la forme `while (b) q`.

$$\Sigma\texttt{(while (b) q,e,m)} = \lim_n \Sigma\texttt{(}p_n\texttt{,e,m)}$$

Remarquons qu'ici les instructions p_i ne sont pas toujours plus courtes que `p`, mais si `p` contient `k` boucles `while` imbriquées alors les p_i en contiennent `k - 1`. La définition de la fonction Σ est donc une définition par récurrence double, d'abord sur le nombre de boucles `while` imbriquées, puis sur la taille de l'instruction.

Exercice 1.5

Quelle est la mémoire Σ`(x = 7 ;,[x = r],[r = 5])` ?

La définition de la fonction Σ de Caml n'est pas très différente de celle de de la fonction Σ de Java. En Caml, n'importe quelle expression dont la valeur est une référence peut figurer à gauche du signe `:=`*, alors qu'en Java, seule une variable peut figurer à gauche du signe* `=`*. La valeur de la fonction Σ de Caml pour l'instruction* `t := u` *se définit ainsi*

– Σ`(t := u,e,m) = m + (`Θ`(t,e,m) =` Θ`(u,e,m))`