8.2.2 Qu'est-ce que « le moi » ?

Si `t` est l'objet construit dans l'exercice 8.6, pour savoir si la réservation est close ou s'il reste des places à l'orchestre, on interprète le terme `t.libre t 0`. En effet, la fonction `t.libre` prend en argument un objet `u` quelconque et un entier `n` et indique si le champ de `u` correspondant à `n` — l'orchestre si `n = 0`, le balcon si `n = 1` — est nul ou non. Autrement dit, la méthode `libre` est *statique*, dans le sens que prend ce mot, par exemple, en Java.

On veut maintenant que la méthode `libre` de l'objet `t` applique à l'objet `t` lui-même, c'est-à-dire qu'on l'invoque en interprétant non le terme `t.libre t 0`, mais plus simplement le terme `t#libre 0`, autrement dit que cette méthode soit *dynamique*.

Une manière de faire cela est de considérer le terme `t#l` comme une simple notation pour le terme `t.l t`. La difficulté est ici que si `t` est un objet et `l` une étiquette de cet objet, alors on ne peut utiliser le terme `t#l` que si le champ `l` est une fonction de type `A -> ...` où `A` est le type de `t` lui-même. Autrement dit, on ne peut utiliser le terme `t#l` que si `l` est l'étiquette d'une méthode. Si `l` est l'étiquette d'un champ qui n'est pas une méthode, il faut continuer d'utiliser la notation `t.l`.

On peut gommer cette différence entre les méthodes et les autres champs en imposant que tous les champs d'un objet soient fonctionnels. Un champ `a` de l'objet `t` dont la valeur est l'entier `3` est donc transformé en un champ dont la valeur est la fonction `fun s -> 3`. Ainsi, le terme `t#a`, c'est-à-dire `t.a t` ou encore `(fun s -> 3) t`, s'interprète en la valeur `3`.

Il est alors raisonnable d'appeler `self` ou `this` la variable liée en premier argument de toutes les méthodes de l'objet. En effet, dans la plupart des langages de programmation, on utilise une variable spéciale `self` ou `this` implicitement liée par la construction de l'objet, qui sert à désigner l'objet lui-même.

Quand toutes les méthodes d'un enregistrement sont des termes de la forme `fun x ->...`, elles s'interprètent sur elles-mêmes, et on peut donc simplifier la règle de sémantique opérationnelle

$$\frac{\texttt{fun } x_1 \texttt{ -> } t_1 \hookrightarrow V_1 \ \ldots}{\{l_1 = \texttt{fun } x_1 \texttt{ -> } t_1,\ \ldots\} \hookrightarrow \{l_1 = V_1,\ \ldots\}}$$

en

$$\frac{}{\{l_1 = \texttt{fun } x_1 \texttt{ -> } t_1,\ \ldots\} \hookrightarrow \{l_1 = \texttt{fun } x_1 \texttt{ -> } t_1,\ \ldots\}}$$

De même, la règle

$$\frac{t \hookrightarrow \{l_1 = V_1,\ \ldots\}}{t.l_i \hookrightarrow V_i}$$

se spécialise en

$$\frac{t \hookrightarrow \{l_1 = \texttt{fun } x_1 \texttt{ -> } t_1,\ \ldots\}}{t.l_i \hookrightarrow \texttt{fun } x_i \texttt{ -> } t_i}$$

et enfin la règle

$$\frac{t \hookrightarrow \{l_1 = V_1,\ \ldots\} \qquad u \hookrightarrow W}{t(l_i \texttt{ <-u }) \hookrightarrow \{l_1 = V_1,\ \ldots,\ l_i = W,\ \ldots\}}$$