```
combiner(f1, f2)
{   remplacer le 2e argument de f1 par f2 et renvoyer f1 }
```

C'est ce qu'on ferait avec un outil comme `yacc`, par exemple.

En fait, si on regarde de plus près, on voit que deux notions sont nécessaires pour formaliser la composition de morceaux de formules : d'une part, une notion de substitution (il faut être capable de requérir le remplacement de toutes les occurrences d'une variable par le même terme), et d'autre part, une notion d'application d'une formule incomplète à une autre.

On peut pour s'en convaincre considérer l'exemple suivant.

(4) a. Un enfant dort
b. [un(enfant)](dort)
c. $[\exists a(Ea \wedge Da)(\text{enfant}(x))]\,(\text{dormir}(y))$

Le λ-calcul est un outil formel extrêmement puissant (c'est-à-dire expressif) qui va précisément nous permettre de spécifier ces différents aspects, d'une manière que l'on peut qualifier de **déclarative**.

3.2 λ-calcul

3.2.1 Introduction

Comme son nom l'indique, le λ-calcul est un **calcul** qui repose sur une unique règle, qu'on peut voir comme une règle de ré-écriture.

Informellement, on définit deux notions/opérations : l'une permet de combiner deux expressions pour en former une nouvelle (application fonctionnelle) ; l'autre permet d'indiquer de façon précise un « point de substitution » (λ-abstraction) :

$$(\lambda x.\alpha x\gamma)A \xrightarrow{\beta} \alpha A\gamma$$

Cette simple règle de calcul donne un grand pouvoir expressif au λ-calcul, équivalent à celui des machines de Turing ou des grammaires de type 0. En particulier, on verra avec les combinateurs que l'on peut trouver des moyens de **coder** sous forme de λ-termes les objets ou notions usuelles (entiers, boucles, booléens, tests...) de telle sorte que l'application de la règle unique (β-réduction) produise les résultats attendus.

3.2.2 Le langage pur (non typé)

Le lambda-calcul a été inventé par Alonzo Church [Church, 1941], pour représenter la notion de fonction mathématique, d'une façon qui évite les problèmes de l'approche basée sur la théorie des ensembles.[1]

3.2.2.1 Syntaxe

Le **vocabulaire** est composé d'un ensemble dénombrable de symboles de variables, de parenthèses, du point, et du symbole λ.

La **syntaxe** est définie par induction :

- Si x est une variable, alors x est un λ-*terme*.
- Si x est une variable, et t un terme, alors $\lambda x.t$ est un λ-*terme* (λ-abstraction)
- Si t_1 et t_2 sont des termes, alors $(t_1)t_2$ est un λ-*terme* (application).

Remarques

- L'application d'un terme φ à un terme ψ est notée $(\varphi)\psi$, à l'inverse de la notation traditionnelle $f(x)$.
- Il n'y a pas de distinction entre variables fonctionnelles et variables arguments. On peut donc former des termes de la forme $\lambda f.(f)f$ (application à soi-même).
- Comme les quantificateurs en logique du premier ordre, l'abstracteur λ est un lieur de variables. On peut définir par conséquent les notions de variables libres et liées : [2]
  (a) La **portée** d'un abstracteur λx dans le terme $\lambda x.\varphi$ est le terme φ.
  (b) Une occurrence d'une variable x dans le terme φ est dite **libre** si elle n'est pas dans la portée d'un abstracteur λx apparaissant dans φ.
  (c) Si $\lambda x.\varphi$ est un sous-terme de ψ et que x est libre dans φ, alors cette occurrence de x est dite **liée** par l'abstracteur λx.

  Exemple : dans le terme $((\lambda x.\lambda y.(x)y)z)x$, la variable x a deux occurrences, l'une libre et l'autre liée.
- La grammaire définissant les λ-termes est non ambigüe, il n'y a qu'une façon de décomposer un terme. On peut définir la notion de sous-terme. Les notations (point et parenthèses) pouvant s'avérer assez lourdes, on adopte les conventions :

$$\begin{aligned} \lambda x_1.\lambda x_2 \ldots \lambda x_n.t &= \lambda x_1 x_2 \ldots x_n.t \\ (\ldots((t)t_1)t_2 \ldots)t_m &= t\,t_1 t_2 \ldots t_m \end{aligned}$$

  Exemple : $\lambda xy.xy$ se lit $\lambda x.\lambda y.(x)y$.
- L'application fonctionnelle est toujours monadique (s'applique à un argument). Toutes les fonctions sont "curryfiées" : une fonction à deux arguments est vue comme une fonction à un argument qui s'applique à un argument.

3.2.2.2 Conversions

Équivalence On définit la relation $\equiv$ entre termes. Intuitivement $t_1 \equiv t_2$ indique que t_1 et t_2 dénotent la même fonction.

Plus rigoureusement, on définit $\equiv$ comme
• une relation d'équivalence entre termes

[1] En particulier, le fait que la notation $3x^2+7$ est ambiguë entre la définition de la fonction qui à x associe $3x^2+7$ et l'application de la fonction à un x particulier.

[2] Version inductive (VL = variables libres) :
- $VL(x) = \{x\}$ (où x est une variable)
- $VL(\ (t_1)t_2\ ) = VL(t_1) \cup VL(t_2)$ (où t_1 et t_2 sont des termes)
- $VL(\lambda x.t) = VL(t) \setminus \{x\}$