UPMC/master/info/4I503 APS
Notes de cours

P. Manoury

Janvier 2017

4 APS3: fonctions et procédures récursives

On complète *APS2* avec la possibilité de donner des définitions de fonctions récursives. En effet, les fermetures définie dans *APS2* pour assigner une valeur aux procédures et fonctions n'offrent pas cette possibilité.

En premier lieu, une expression fonctionnelle, qui est une fonction anonyme, ne peut être récursive, puisqu'elle ne dispose pas de nom pour s'invoquer elle-même. Mais quand bien même on nommerait l'expression fonctionnelle avec la déclaration `CONST`, la discipline de liaison statique rend inopérante l'auto référence. En effet la fermeture associée à l'exprssion fonctionnelle `[`x`:`t`]e` prend la forme $\langle e, r + x\rangle$ où r est l'environnement contemporain de la définition. C'est elle qui est activée lors des applications de la fonction pour obtenir la valeur de e avec l'environnement $r + x$ actualisé. Si e fait usage du nom de la fonction dont elle est le corps, celui-ci ne figure pas dans l'environnement.

Pour obtenir des définitions récursives, il faut introduire un peu de dynamicité dans les fermetures.

4.1 Syntaxe

Outre l'ajout du trait de syntaxe spécifique aux définitions récursives, nous ajouterons également de défintion des fonctions au moyen d'expressions fonctionnelles à la clause de définition `FUN`

Lexique on rajoute le mot clef `REC`

Grammaire on étend les possibilités de déclarations

DEC ::= ...
`FUN` ident TYPE `[` ARGS `]` EXPR
`FUN REC` ident TYPE `[` ARGS `]` EXPR
`FUN REC` ident TYPE `[` ARGS `]` CMDS
`PROC REC` ident `[` ARGS `]` CMDS

4.2 Typage

La règle de typage pour les définitions de fonctions par une expression fonctionnelle est similaire à celle des définitions de constantes.

(FUN) si $G; x_1 : t_1; \ldots; x_n : t_n \vdash_{\text{EXPR}} e : t$ et si $G \vdash_{\text{CMDS}} cs : t'$ alors $G \vdash_{\text{CMDS}}$ (`FUN` $x\ t$ `[`x_1`:`$t_1, \ldots, x_n$`:`t_n`]`$e; cs$) : t'

Le corps d'une fonction ou procédure récursive se mentionnant elle-même, on ajoute son assignation de type au contexte de typage.

(FRECF) si $G; x_1 : t_1; \ldots; x_n : t_n; x : (t_1 * \ldots * t_n)\text{->}t \vdash_{\text{EXPR}} e : t$ et si $G \vdash_{\text{CMDS}} cs : t'$ alors
$G \vdash_{\text{CMDS}}$ (`FUN REC` $x\ t$ `[`x_1`:`$t_1, \ldots, x_n$`:`t_n`]`$e; cs$) : t'

(FRECP) si $G; x_1 : t_1; \ldots; x_n : t_n; x : (t_1 * \ldots * t_n)\texttt{->}t \vdash_{\text{CMDS}} cs : t$, et si $G \vdash_{\text{CMDS}} cs' : t'$ alors
$G \vdash_{\text{CMDS}} (\texttt{FUN REC}\ x\ t\ \texttt{[}x_1\texttt{:}t_1, \ldots, x_n\texttt{:}t_n\texttt{]}\ \texttt{[}cs\texttt{]}; cs') : t'$

(PREC) si $G; x_1 : t_1; \ldots; x_n : t_n; x : (t_1 * \ldots * t_n)\texttt{->void} \vdash_{\text{CMDS}} cs : \mathsf{void}$, et si $G \vdash_{\text{CMDS}} cs' : t'$ alors
$G \vdash_{\text{CMDS}} (\texttt{PROC REC}\ x\ t\ \texttt{[}x_1\texttt{:}t_1, \ldots, x_n\texttt{:}t_n\texttt{]}\ \texttt{[}cs\texttt{]}; cs') : t'$

4.3 Sémantique opérationnelle

L'environnement capturé par une femerture doit pouvoir recevoir la valeur de la fonction qu'elle représente, c'est-à-dire, sa propre valeur associée au nom de la fonction qu'elle représente. Une *fermeture récursive* s'écrit $\langle e, r - x + [x_1, \ldots, x_n]\rangle$ où x est le nom de la fonction récursive. Nous utilisons le signe $-$ (moins) pour distinguer ce nom.

Déclarations On ajoute aux régles sémantiques pour les suites de commandes celles correspondant à nos nouvelles possibilités de définitions.

(FUN) si $(r[x = \langle e, r + [x_1, \ldots, x_n]\rangle], m \vdash_{\text{CMDS}} cs \rightsquigarrow (v, m')$ alors $r, m \vdash_{\text{CMDS}} (\texttt{FUN}\ x\ t\ \texttt{[}x_1\texttt{,}\ldots\texttt{,}x_n\texttt{]}e\texttt{;}cs) \rightsquigarrow (v, m')$

(FRECF) si $(r[x = \langle e, r - x + [x_1, \ldots, x_n]\rangle], m \vdash_{\text{CMDS}} cs \rightsquigarrow (v, m')$ alors
$r, m \vdash_{\text{CMDS}} (\texttt{FUN REC}\ x\ t\ \texttt{[}x_1\texttt{,}\ldots\texttt{,}x_n\texttt{]}e\texttt{;}cs) \rightsquigarrow (v, m')$

(FRECP) si $(r[x = \langle cs, r - x + [x_1, \ldots, x_n]\rangle], m \vdash_{\text{CMDS}} cs \rightsquigarrow (v, m')$ alors
$r, m \vdash_{\text{CMDS}} (\texttt{FUN REC}\ x\ t\ \texttt{[}x_1\texttt{,}\ldots\texttt{,}x_n\texttt{]}\ \texttt{[}cs\texttt{]}\texttt{;}cs) \rightsquigarrow (v, m')$

(PREC) si $(r[x = \langle e, r - x + [x_1, \ldots, x_n]\rangle], m \vdash_{\text{CMDS}} cs \rightsquigarrow (v, m')$ alors
$r, m \vdash_{\text{CMDS}} (\texttt{PROC REC}\ x\ \texttt{[}x_1\texttt{,}\ldots\texttt{,}x_n\texttt{]}e\texttt{;}cs) \rightsquigarrow (v, m')$

Application L'application d'une fermeture récursive demande un traitement particulier. On ajoute la règle suivante:

(APPR) Si $r, m \vdash_{\text{EXPR}} e \rightsquigarrow \langle e', r' - x + [x_1, \ldots, x_n]\rangle$, si $r, m \vdash_{\text{EXPR}} e_1 \rightsquigarrow v_1$ et $\ldots$ et $r, m \vdash_{\text{EXPR}} e_1 \rightsquigarrow v_1$ et si $r'[x = \langle e', r' - x + [x_1, \ldots, x_n]\rangle, x_1 = v_1, \ldots, x_n = v_n], m \vdash_{\text{EXPR}} e' \rightsquigarrow v$ alors $r, m \vdash_{\text{EXPR}} (e\ e_1 \ldots e_n) \rightsquigarrow v$.

4.4 Sémantique dénotationnelle

La définition d'une fonction comme une expression fonctionnelle se traite de la même manière que celle des déclarations de constantes. On doit normalement avoir ici une expression fonctionnelle dont la valeur sera une fermeture.

$$\mathbf{Cs}[[\texttt{FUN}\ x\ t\ e;\ cs]]\rho\sigma\kappa \quad = \quad \mathbf{E}[[e]]\rho\sigma(\lambda(v, s).\mathbf{Cs}[[cs]](\rho[x = inF(v)])\ s\ \kappa$$

Les définitions récursives produisent des *fermetures récursives*. Ce qui se réalise aisément en sémantiques dénotationnelle avec le combinateur de point fixe.

$\mathbf{Cs}[[\texttt{FUN REC}\ x\ \tau\ \texttt{[}x_1 : t_1\texttt{]}e;\ cs]]\rho\sigma\kappa$
$=$ let $f = !z\lambda v_1\lambda s.\lambda k.\mathbf{E}[[e]](\rho[x = inF(z), x_1 = inV(v_1)])\ s\ k$ in
$\mathbf{Cs}[[cs]](\rho[x = inF(f)])\sigma\kappa$

$\mathbf{Cs}[[\texttt{FUN REC}\ x\ \tau\ \texttt{[}x_1 : t_1\texttt{]}cs;\ cs']]\rho\sigma\kappa$
$=$ let $f = !z\lambda v_1\lambda s.\lambda k.\mathbf{Cs}[[cs]](\rho[x = inF(z), \texttt{*ret*} = inC(\kappa), x_1 = inV(v_1)])\ s\ k$ in
$\mathbf{Cs}[[cs']](\rho[x = inF(f)])\sigma\kappa$

$\mathbf{Cs}[[\texttt{PROC REC}\ x\ \tau\ \texttt{[}x_1 : t_1\texttt{]}cs;\ cs']]\rho\sigma\kappa$
$=$ let $f = !z\lambda v_1\lambda s.\lambda k.\mathbf{Cs}[[cs]](\rho[x = inF(z), x_1 = inV(v_1)])\ s\ k$ in
$\mathbf{Cs}[[cs']](\rho[x = inF(f)])\sigma\kappa$