LANGAGES FONCTIONNELS

Tels ML Miranda

- Utilisation de fonctions ... comme moyen principal de former des boucles
  donc nécessité de traduction en code efficace

- Variables libres: comme en algol-60 etc.

- Possibilité d'une fonction dont le type du résultat est une fonction

- **purement** fonctionnel: aucun effet de bord

- définition par motif:
  $f(0, y) = y; f(x, 0) = x; f(x, y) = x * y$

- définition de structures par leurs fonctions de construction:
  $$\begin{aligned} arbreint ::= &\ feuille\ int| \\ &\ branche(arbreint)(arbreint) \end{aligned}$$

LES REGLES DE TYPE

Typage fort (strict)
Impossible d'appliquer un objet dans un contexte interdit
condition testée par le compilateur

- types de base

- listes $[a, b, c]$ ou $(h : t)$, n-uplets (a, b, c)

- fonctions $T1 \rightarrow T2$ est le type d'une fonction à un paramètre de type $T1$ renvoyant un résultat de type $T2$

- Complètement récursif: les éléments des listes, le paramètre d'une fonction et le résultat d'une fonction peuvent être des fonctions

TYPAGE IMPLICITE

Pas d'obligation de déclarer les types! Le compilateur peut (doit) calculer quels sont les types de chaque (identificateur, expression)

$f\ x\ =\ x{+}1$; x est $entier$ et f est $entier\ \rightarrow\ entier$

$f\ [\,]\ =\ 0;\ f(h:t)\ =\ h$; f est $[entier]\ \rightarrow\ entier$

$f\ [\,]\ L2\ =\ [\,];\ f\ L1\ [\,]\ =\ [\,];$
$\quad f\ (h1:t1)\ (h2:t2)\ =\ (h1+h2:f\ t1\ t2)$
f est $[entier]\ \rightarrow\ [entier]\ \rightarrow\ [entier]$

($X\ \rightarrow\ Y\ \rightarrow\ Z$ équivaut à $X\ \rightarrow\ (Y\ \rightarrow\ Z)$ est à peu près le type d'une fonction de deux paramètres)

POLYMORPHISME

Un objet peut avoir plusieurs types possibles; le bon type est choisi (par le compilateur) selon le contexte:

$l\ [\,]\ =\ 0;\ l\ (h : t)\ =\ 1 + l\ t$
l (longueur) peut être $[entier]\ \rightarrow\ entier$ ou $[char]\ \rightarrow\ entier$ etc.

On écrit le type de l: $[\alpha]\ \rightarrow\ entier$

Un deuxième exemple: concaténation:
$conc\ [\,]\ L\ =\ L;\ conc\ (h : t)\ L\ =\ (h : conc\ t\ L)$
avec type $[\alpha]\ \rightarrow\ [\alpha]\ \rightarrow\ [\alpha]$

(A noter que tout α devra être substitué par le même type)

$paire\ [\,]\ [\,]\ =\ [\,];$
$\quad paire\ (h1 : t1)\ (h2 : t2)\ =\ ((h1, h2) :$

$paire\ t1\ t2$)
avec type $[\alpha]\ \rightarrow\ [\beta]\ \rightarrow\ [(\alpha, \beta)]$

(Aussi des constructeurs polymorphes de types;
$arbre\ int$, $arbre\ char$, etc.)

COMMENT DECIDER LES TYPES?

Pour décider le type d'une fonction définie (f):

- Décider les types de toutes les définitions locales

- Pour chaque expression et sous-expression dans la définition allouer un identificateur distinct de type ($T0$, $T1$, $T2 \cdots$)

- écrire des équations pour les contraintes sur ces types

- Si une sous-expression est un identificateur déjà défini, son équation est $Ti \quad =$ type déjà décidé (avec de nouveaux identificateurs pour ses types génériques $(\alpha \cdots)$)

- Chaque occurrence de f a le même type

- Equations spécifiques pour les opérateurs standard

- Dans une liste tous les éléments ont le même type et le type de la liste est [ce type]

- Le type d'un n-uplet est le n-uplet des types des éléments

- Quand une fonction (de type Ti) est appliquée à un paramètre (de type Tj) et donne une résultat de type Tk on a
$Ti \; = \; Tj \; \rightarrow \; Tk$

- On “résout” toutes ces équations

- (S’il y a une solution) les nouveaux identificateurs de type qui reste sont les types génériques du type de f.

RESOLUTION DES EQUATIONS

Il faut trouver (si possible) une *substitution* (fonction des identificateurs de type aux expressions de type) qui vérifie toutes les équations

On prend les équations dans n'importe quel ordre et résout chacun en rajoutant à la substitution les informations trouvées

Si un coté d'une équation est une variable, on rajoute à la substitution que cette variable donne le type de l'autre coté

Si les deux types sont construits par le même constructeur de types (par exemple $liste$ ou

$fonction$), on rajoute des équations pour les types utilisés dans la construction

Sinon erreur

Terminaison garantie

EVALUATION PARESSEUSE

Le mode d'évaluation où on n'évalue un paramètre que quand (si) la fonction dépend de sa valeur et on n'évalue que les parties d'une structure dont dépend la fonction

Exemple:
$sommepremiers\ (h1 : t1)\ (h2 : t2) \ = \ h1 + h2$
peut être évalué sans évaluer $t1$ et $t2$

Permet même la manipulation (raisonnable) de structures infinies!
$puissances2\ i \ = \ (i : puissances2\ 2 * i)$
$sommepremiers\ (puissances2\ 4)\ (puissances2\ 16)$
calcule 20.

Evaluation paresseuse est standard en Miranda et d'autres langages.