A FAMILY OF RECURSIVE SCHEMAS

Pascal MANOURY
Université Pierre et Marie Curie
Paris France
pascal.manoury@lip6.fr

Marianne SIMONOT
Conservatoire National des Arts et Métiers
Paris France
simonot@cnam.fr

Introduction

The early computer scientists were mathematicians concerned with the formalization of *computability*. They proposed several definitions of this concept (see, for instance [1]) The modern computer scientists should pay attention to the classical approach of computability using the definition of *recursive functions* for two reasons.

A lot of recursive functions *are* programs : one can easily rephrase the (recursive) equations defining a recursive function as an ML program.

A lot of programs become understandable when we can exhibit the recursive function they compute.

The first step in theoretical investigation of recursive functions was to define the set of *primitive recursive functions*. The main feature of this definition is to validate as belonging to the set of primitive recursive functions the one defined according to a *primitive recursive schema*. Then one have a syntactical criterion for checking the correctness of the definitions. And therefore, checking correctness can be automatized. Moreover the equations defining the primitive recursive schema give an *algorithm* to compute the function. But this first step has two limitations.

One can define primitive recursive functions without following the definition schema. The definition of primitive recursive functions is done in an *extensional* way while the computer scientist should be also interested by an *intensional* characterization.

There exist computable functions which are not primitive recursive. Then the set of computable functions is wider than the one of primitive recursive functions. The improvement proposed by mathematicians is to extend the set of recursive function with a new definition schema : the *minimization schema*. But ones more, one can define a non primitive recursive function by equations rather than using the minimization schema.

We think that if recursive functions are a key concept for foundation of computing, it is not only from an extensional point of view but also from an intentional or operational one. An equational specification denotes an algorithm computing the function. From this point of view the usual schemas are not sufficient. We propose in this paper to enrich the admissible definition schemas in a way that preserves the algorithmic content of the specifications.

In the following, we describe a family of definition schemas (section 1). Their design is an extension of the usual equational definitions involving second order simply typed λ-terms. In section 2, we describe an algorithm which takes as input an usual specification of functions and produces as output either a failure or an instance of a schema which is equivalent from an operational point of view. This algorithm can then be viewed as a decision procedure for termination. In section 3 we propose, as a conjecture, an extensionnal property of our schemas.

1 An overview of the schemas family

Our family can be figured as the matrix $\begin{array}{cccc} \mathcal{S}_1^0 & \dots & \mathcal{S}_1^k & \dots \\ \vdots & \ddots & \vdots & \\ \mathcal{S}_r^0 & \dots & \mathcal{S}_r^k & \dots \\ \vdots & & \vdots & \ddots \end{array}$

Roughly speaking $\mathcal{S}_r^k$ represents the class of functions defined with r recursions and k nestings of recursive calls. For instance :

1. in $\mathcal{S}_1^0$ we have the primitive recursive schema $f(0,m) = g_1(m)$ and $f(n+1,m) = g_2(n,m,f(n,m))$;
2. in $\mathcal{S}_1^1$ we have the function defined by $f(0,m) = m$ and $f(n+1,m) = f(n,f(n,m))$;
3. in $\mathcal{S}_2^0$ we have the function defined by the double recursion $f(0,m) = m$, $f(n+1,0) = n+1$ and $f(m+1,n+1) = f(n,m+1) + f(n+1,m)$
4. in $\mathcal{S}_2^1$ we have the Ackermann's function.

The above examples can be reformulated as :

1. $f(0,m) = (\lambda y.g_1(y))(m)$ and $f(n+1,m) = (\lambda x\lambda y\lambda f.g_2(x,y,f(y)))(n,m,\lambda y.f(n,y))$;
2. $f(0,m) = (\lambda y.y)(m)$ and $f(n+1,m) = (\lambda x\lambda y\lambda f.f(f(y)))(n,m,\lambda y.f(n,y))$
3. $f(0,m) = (\lambda x_2.x_2)(m)$,
   $f(n+1,0) = (\lambda x_1\lambda f_1.x_1+1)(n,\lambda x_2.f(n,x_2))$,
   $f(n+1,m+1) = (\lambda x_1\lambda x_2\lambda f_1\lambda f_2.f_1(x_2)+f_2)(n,m,\lambda x_2.f(n,x_2),f(n+1,m))$;
4. the Ackermann's function is
   $f(0,m) = (\lambda x_2.x_2+1)(m)$,
   $f(n+1,0) = (\lambda x_1\lambda f_1.f_1(1))(n,\lambda x_2.f(n,x_2))$,
   $f(n+1,m+1) = (\lambda x_1\lambda x_2\lambda f_1\lambda f_2.f_1(f_2))(n,m,\lambda x_2.f(n,x_2),f(n+1,m))$.

So, each schema $\mathcal{S}_j^i$ takes the form of an equational system : $\begin{cases} f(t_1^1,\dots,t_n^1) = U_1 \\ \vdots \\ f(t_1^m,\dots,t_n^m) = U_m \end{cases}$
where the t_j^i are either a variable, either 0, either a term $n+1$ where n is a variable. Moreover, the system is left linear, orthogonal and complete with respect to pattern matching over tuples.
For simplicity, we assume that all our functions have only one parameter, say y. The *right hand sides* have form : $H(x_1,..,x_k,y,\varphi_1,...,\varphi_k)$, where : the x_i's and y are variables from the left hand side; H is a simply typed λ-combinator of the form $\lambda\vec{x}.t$ where t contains only applications of already defined symbols or bound variables and for all $i \in [1..k]$ φ_i has the form $\lambda z_1..\lambda z_{n-i}.f(v_1,..,v_n)$ where the v_j's are either 0 either x_j+1, with $(1 \leq j < i)$ either x_i or one of the bounded z_j's and f is the defined function. As a consequence, H is a second order combinator.

The combinators To define the $\mathcal{S}_j^i$ family, we first define the set $\mathcal{H}$ of allowed H combinators. Let $r \geq 1$, we define the set $\mathcal{H}_r$ of second order simply typed combinators as the set of terms $\lambda x_1..\lambda x_r\lambda y\lambda f_1..\lambda f_r.t$. For $i \in [1..r]$, f_i has type $o^{r-i}, o \to o$. The term t is of type o and contains only applications of already defined functions or bounded variables.

We define a hierarchy inside each $\mathcal{H}_r$: we write $\mathcal{H}_r^i$ for the subset of $\mathcal{H}_r$ such that t contains i nestings of functional variables f_j.

The schemas Let's now define $\mathcal{S}_r$ by induction on r.

The set $\mathcal{S}_1$ is the set of definition schemas $\left\{\begin{array}{lcl} f(0,y) & = & g(y) \\ f(x+1,y) & = & H(x,y,\lambda y.f(x,y)) \end{array}\right.$ with $H \in \mathcal{H}_1$.

If $\varphi = \lambda z_1..\lambda z_r.f(v_1,..,v_n)$, and $j \in [1..n]$, we write $\varphi^{\langle 0\rangle}$ for $\lambda z_1..\lambda z_r.f(v_1,..,v_j,0,v_{j+1},..,v_n)$ and $\varphi^{\langle +\rangle}$ for $\lambda z_1..\lambda z_r.f(v_1,..v_j,x_0,v_{j+1},..,v_n)$ where x_0 is a new variable. Let $U = H(x_1,..,x_r,y,\varphi_1,...,\varphi_r)$ be a right hand side with $H \in \mathcal{H}_r$, we write $U^{\langle 0\rangle}$ for $H(x_1,..,x_r,y,\varphi_1^{\langle 0\rangle},...,\varphi_r^{\langle 0\rangle})$ and $U^{\langle +\rangle}$ for $H'(x_0,x_1,..,x_r,y,\varphi_1^{\langle +\rangle},...,\varphi_r^{\langle +\rangle})$ where $H' \in \mathcal{H}_{r+1}$.

Let $\left\{\begin{array}{l} f(t_1^1,\dots,t_n^1) = U_1 \\ \vdots \\ f(t_1^m,\dots,t_n^m) = U_m \end{array}\right.$ and $\left\{\begin{array}{l} f(v_1^1,\dots,v_n^1) = W_1 \\ \vdots \\ f(v_1^p,\dots,v_n^p) = W_p \end{array}\right.$ be two definition schemas belonging respectively to $\mathcal{S}_r$ and $\mathcal{S}_l$.

The schema defined as

$$\left\{\begin{array}{lcl} f(t_1^1,\dots,t_j^1,0,t_{j+1}^1,\dots,t_n^1) & = & U_1^{\langle 0\rangle} \\ \vdots & & \\ f(t_1^m,\dots,t_j^m,0,t_{j+1}^m,\dots,t_n^m) & = & U_m^{\langle 0\rangle} \\ f(v_1^1,\dots,v_j^1,x_0+1,v_{j+1}^1,\dots,v_n^1) & = & W_1^{\langle +\rangle} \\ \vdots & & \\ f(v_1^p,\dots,v_j^p,x_0+1,v_{j+1}^p,\dots,v_n^p) & = & W_p^{\langle +\rangle} \end{array}\right.$$

belongs to $\mathcal{S}_{max(l,r)+1}$.

As we did for the set of combinators, we divide each $\mathcal{S}_r$ according to the nesting of inductive calls. We write $\mathcal{S}_r^i$ for the subset of $\mathcal{S}_r$ such that i is the greatest index such that there exists a right hand side $H(x_1,..,x_s,y,\varphi_1,..\varphi_s)$ with $H \in \mathcal{H}_s^i$ and $s \leq r$.

2 Decision procedure

We present now how our classification of definition schemas can be used to check the correctness of an usual first order definition of a recursive function.

Let $\left\{\begin{array}{l} f(t_1^1,\dots,t_n^1) = u_1 \\ \vdots \\ f(t_1^m,\dots,t_n^m) = u_m \end{array}\right.$

be a recursive definition where $u_1,..,u_m$ are first order terms. Our method is the following :

1. from the tuples $(t_1^1,\dots,t_n^1),..,(t_1^m,\dots,t_n^m)$ we build a set of possible right hand sides, say $X_1(\vec{x}_1,\vec{y}_1,\vec{\varphi}_1)$, $..,X_m(\vec{x}_m,\vec{y}_m,\vec{\varphi}_m)$ where $X_1,..,X_m$ are functional variables;
2. then we solve the second order matching problem $X_1(\vec{x}_1,\vec{y}_1,\vec{\varphi}_1) = u_1,..,X_m(\vec{x}_m,\vec{y}_m,\vec{\varphi}_m) = u_m$ where $X_1,..,X_m$ are the unknowns.

Step 1 is very closed to the construction of what we called a *distributing tree* in [2]. A distributing tree for the previous definition of f is a tree whose nodes are labeled with substitutions on the set V of variables $x_1,\dots x_n$ such that :

1. the root is labeled with the identity function ;
2. to each left hand side $f(t_1^i,\dots,t_n^i)$ of the definition, there exists exactly one leaf labeled with σ_j such that $\sigma_j(f(x_1,\dots,x_n)) = f(t_1^i,\dots,t_n^i)$;

3. each node σ possesses exactly two children labeled with σ_1 and σ_2 such that : $\sigma_1 - \sigma = \{(v, 0)\}$ and $\sigma_2 - \sigma = \{(v, v+1)\}$ where v is a variable of $\sigma(f(x_1, \dots, x_n))$.

We build the possible right hand side for $f(t_1^i, \dots, t_n^i)$ with the help of the distributing tree as follows : let $C = \sigma_0, \sigma_1, \dots, \sigma_k$ be the branch of the distributing tree such that $\sigma_k(f(x_1, \dots, x_n)) = f(t_1^i, \dots, t_n^i)$.
By definition, for all i $(0 < i \leq k)$ $\sigma_i - \sigma_{i-1} = \{(v, t)\}$.
Let $V(\sigma_i)$ denotes v and $T(\sigma_i)$ denotes t, let $B(\sigma_i)$ be the subset of V such that $\sigma_i(x) = x$.
Let $\phi(\sigma_i)$ be the term $\lambda \vec{z}.\sigma_{i-1}(f(x_1, \dots, x_n))$ where $\vec{z} = B(\sigma_i)$ and $C' = \sigma'_1, \dots, \sigma'_p$ $(p \leq k)$ be the subsequence of C such that $T(\sigma_i) \neq 0$.
The possible right hand side is : $X(\vec{x}, \vec{y}, \vec{\phi})$ where :

- $\vec{x} = V(\sigma'_1) \dots V(\sigma'_p)$
- $\vec{y} = B(\sigma_k)$
- $\vec{\phi} = \phi(\sigma'_1) \dots \phi(\sigma'_p)$

For step 2, we can use a second order pattern matching algorithm for simply typed λ-calculus (see [3]) with the restriction that we do not allow to use the defined function symbol f to perform *imitation*.

3 Extensional property : a conjecture

Classical presentations of recursive functions introduces several definitions schemas which are proved to define, in fact, primitive recursive functions. This is, in general, done by reducing the original function definition to a new one matching the primitive recursive schema. The method is based on the possibility of encoding tuples as natural numbers. For instance, in [1], one can find that simple recursion with substitutions for parameters are primitive recursive ; simple recursion with one nesting of recursive call remains primitive recursive ; unested double recursions are primitive recursive and so on. In our classification, those three examples correspond respectively to definition schemas belonging to $\mathcal{S}_1^0$, $\mathcal{S}_1^1$ and $\mathcal{S}_2^0$. As we have that the primitive recursive schema is in $\mathcal{S}_1^0$, we have that functions defined in $\mathcal{S}_1^1$ and $\mathcal{S}_2^0$ have also a definition in $\mathcal{S}_1^0$. This result may be generalized to the following : for all $i > 0$ and $j > 1$ any function defined in $\mathcal{S}_1^i$ or $\mathcal{S}_j^0$ can be defined in $\mathcal{S}_1^0$. And doing one more generalization step, we conjecture the interesting property of our matrix that for any $\mathcal{S}_{i+1}^i$ of its diagonal, for any $j > i$, $\mathcal{S}_{i+1}^j$ and $\mathcal{S}_{j+1}^i$ can be extensionally reduced to $\mathcal{S}_{i+1}^i$. Moreover, as each $\mathcal{S}_{i+1}^i$ contains the *ith* function of the *Ackermann's family*, this reduction is maximal. Intuitively, the matrix can extensionaly be reduced to its diagonal.

Conclusion

The definition schemas family we introduced in this paper allow to check syntactically the correctness of recursive definitions. This is of interest for computer scientists involved in the development of theorem provers or proof assistant systems where this correctness proof can be automatized. Moreover, if we restricted here our presentation to recursive functions over naturals whose correctness proof needs simple structural induction, it is obvious that our schemas can be extended to any king of data types expressed as term algebras and that we can support some kind of *structural course of value induction* as the one of Fibonacci's sequence, for instance. We think also that our approach can be a tool for the investigation of the computational behavior of functional algorithms.

References

[1] R. PÉTER *Recursive functions* Academic Press, New York, 1967

[2] P. MANOURY, M. SIMONOT *Automatizing termination proofs of recursively defined functions* TCS 135, 1995

[3] G. DOWEK *A second order pattern matching algorithm in the cube of typed λ-calculi* LNCS 520, 1991