SCHEMES WITH RECURSION ON HIGHER TYPES

Klaus Indermark

Lehrstuhl für Informatik II, RWTH Aachen

Büchel 29 - 31, D - 51oo Aachen

1. Introduction

In program scheme theory it has been shown that recursion is a more powerful control structure than iteration: flowchart schemes are translatable into recursion schemes, but not vice versa. For a comprehensive treatment of this and related results see the books of ENGELFRIET |2| and GREIBACH |4|. These recursion schemes do not involve objects of higher functional types. In order to investigate the impact of higher types on the definition of lower type objects we introduce a class of program schemes which are based on typed combinators including fixed-point combinators.

These "combinator schemes" have two characteristic features:

1) Cartesian products are not eliminated by means of SCHÖNFINKEL's isomorphisms $((A \times B) \rightarrow C) \cong (A \rightarrow (B \rightarrow C))$ because this would lead to irrelevant higher types.

2) The schemes are completely uninterpreted in the sense of NIVAT |6|: <u>if</u> ... <u>then</u> ... <u>else</u> ... and predicate symbols do not occur; operation symbols will be interpreted as continuous operations on chain-complete partially ordered sets.

In this paper we define syntax and semantics of combinator schemes which originated from the author's lectures on recursive definitions |5|. Furthermore, we reduce this class of schemes to certain subclasses. Due to space limits proofs are omitted and will be published elsewhere.

2. Cartesian types

Types are names for sets. Starting from a set of base types we generate cartesian types by use of two binary operations. They will be interpreted as cartesian product and exponentiation.

For a given set I of base types we define the set $\underline{typ}(I)$ of cartesian types over I as the smallest class $K \subseteq (I \cup \{[\,,\,],\ |\,,e\})^+$ such that

(i) $\{e\} \cup I \subseteq K$

(ii) $t_1,t_2 \in K \Rightarrow [t_1 \mid t_2] \in K$

(iii) $t_1,t_2 \in K \setminus \{e\} \Rightarrow t_1 t_2 \in K$.

$\underline{typ}$ (I) can be considered as the carrier of an algebra with operations $(t_1,t_2) \mapsto [t_1 \mid t_2]$, $(t_1,t_2) \mapsto t_1 t_2$ with unit e and $(\ ) \mapsto e$. Moreover, this algebra is freely generated by I in the class of all algebras with two binary and one nullary operation where in addition one binary operation is associative with the nullary operation as its neutral element.

3. Typed combinators

Typed combinators are names for canonical elements of certain function spaces.

The set $\Gamma(I)$ of combinators over I is defined as the disjoint union of the following symbol sets:

(i) composition combinators $\{\underline{C\ \langle t_1,t_2,t_3\rangle} \mid t_\nu \in \underline{typ}(I)\}$

(ii) tupling combinators $\{\underline{T\ \langle t_1,t_2,t_3\rangle} \mid t_\nu \in \underline{typ}(I)\}$

(iii) abstraction combinators $\{\underline{K\ \langle t_1,t_2\rangle} \mid t_\nu \in \underline{typ}(I)\}$

(iv) Schönfinkel combinators $\{\underline{S\ \langle t_1,t_2,t_3\rangle} \mid t_\nu \in \underline{typ}(I)\}$

and $\{\underline{\$\ \langle t_1,t_2,t_3\rangle} \mid t_\nu \in \underline{typ}(I)\}$

(v) fixed-point combinators $\{\underline{Y\ \langle t\rangle} \mid t \in \underline{typ}(I)\}$

Their meaning can be guessed from the notation and will be defined below. Since we deal with cartesian products we have included tupling and Schönfinkel combinators. Fixed-point combinators are required because infinite types, and therefore self-application, lay outside the scope of this paper.

According to its indices each combinator over I is given a type.

The mapping $\tau : \Gamma(I) \to \underline{typ}(I)$ is defined by

(i) $\underline{C\ \langle r,s,t\rangle} \mapsto [[r \mid s][s \mid t] \mid [r \mid t]]$

(ii) $\underline{T\ \langle r,s,t\rangle} \mapsto [[r \mid s][r \mid t] \mid [r \mid st]]$

(iii) $\underline{K\ \langle s,t\rangle} \mapsto [t \mid [s \mid t]]$

(iv) $\underline{S\ \langle r,s,t\rangle} \mapsto [[rs \mid t] \mid [r \mid [s \mid t]]]$

(v) $\underline{\$\ \langle r,s,t\rangle} \mapsto [[r \mid [s \mid t]] \mid [rs \mid t]]$

(vi) $\underline{Y\ \langle t\rangle} \mapsto [[t \mid t] \mid t]$

Syntactically, these types allow for typed combinations of combinators and operation symbols, whereas semantically they will indicate the corresponding function spaces.

4. Combinator schemes

Now, we can introduce the class of combinator schemes over an alphabet Ω of heterogeneous operation symbols as the typed combination closure of Ω and Γ. To be more precise let $\Omega(I)$ be a set of <u>operation symbols over I</u> typed by
$\tau : \Omega(I) \to [I^* \mid I]$.
The class $\underline{rec}(\Omega(I))$ of <u>combinator schemes over $\Omega(I)$</u> is defined as the smallest subclass

$$K \subseteq (\Omega(I) \cup \Gamma(I) \cup \{(\,,\,)\})^+ \quad \text{typed by} \quad \tau : K \to \underline{typ}(I) \quad \text{such that}$$

(i) $\Omega \cup \Gamma \subseteq K$

(ii) $S, S_1, \ldots, S_n \in K,\ \ \tau(S) = [t_1 t_2 \ldots t_n \mid t],\ \ \tau(S_\nu) = t_\nu,$
$1 \leqslant \nu \leqslant n \implies (S\, S_1\, S_2 \ldots S_n) \in K$ with type t

The notation <u>rec</u> stands for "recursion". In fact, the fixed-point combinators will describe recursion on higher types.

5. Semantics

Types will be interpreted as chain-complete posets (cpo's), operation symbols as continuous operations, combinators as continuous operators, and combinations as applications.

The class <u>cpo</u> of all cpo's has the following property. For $A,B \in \underline{cpo}$ we know that the set $(A \to B)$ of continuous functions from A to B w.r.t. point-wise ordering and the associative cartesian product $A \times B$ w.r.t. component-wise ordering belong again to <u>cpo</u>. By $\perp$ we denote the one-element cpo $\{\emptyset\}$ which is the neutral element of $\times$. Thereby, we can define for a given class $\mathfrak{K}$ of cpo's its <u>cartesian closure</u> $\underline{\text{cart}}(\mathfrak{K})$ as the smallest subclass of $\underline{\text{cpo}}$ that contains $\mathfrak{K} \cup \{\underline{\bot}\}$ and that is closed under exponentiation $(A,B) \mapsto (A \to B)$ and product $(A,B) \mapsto A \times B$.

Let $A := \{A^i \mid i \in I\} \subseteq \underline{\text{cpo}}$. A is called a <u>(continuous) interpretation of I</u> and it extends uniquely to all cartesian types over I:

The mapping $h_A : \underline{\text{typ}}(I) \to \underline{\text{cart}}(A)$ is determined by

$$\begin{aligned} i &\mapsto A^i \\ e &\mapsto \underline{\bot} \\ [s \mid t] &\mapsto (h_A(s) \to h_A(t)) \\ st &\mapsto h_A(s) \times h_A(t) \end{aligned}$$

For the uniqueness of h_A see the remarks in 2. on the algebraic character of $\underline{\text{typ}}(I)$. They prove h_A to be an algebra homomorphism.

Notation: We write A^t for $h_A(t)$ as in the case of base types and $\underline{\text{obj}}(A) := \{F \in A^t \mid t \in \underline{\text{typ}}(I)\}$.

An interpretation A of I implies the semantics of the typed combinators: The mapping $[\![\ ]\!]_A : \Gamma(I) \to \underline{\text{obj}}(A)$ is given by

$$\begin{aligned}
&[\![C\langle r,s,t\rangle]\!]_A : A^{[r|s]} \times A^{[s|t]} \to A^{[r|t]}, && (f,g) \mapsto \lambda x.\ g(f(x)) \\
&[\![T\langle r,s,t\rangle]\!]_A : A^{[r|s]} \times A^{[r|t]} \to A^{[r|st]}, && (f,g) \mapsto \lambda x.\ (f(x),\ g(x)) \\
&[\![K\langle s,t\rangle]\!]_A : A^t \to A^{[s|t]}, && f \mapsto \lambda x.\ f \\
&[\![S\langle r,s,t\rangle]\!]_A : A^{[rs|t]} \to A^{[r|[s|t]]}, && f \mapsto \lambda x.\ \lambda y.\ f(x,y) \\
&[\![\$\langle r,s,t\rangle]\!]_A : A^{[r|[s|t]]} \to A^{[rs|t]}, && f \mapsto \lambda(x,y).\ f(x)(y) \\
&[\![Y\langle t\rangle]\!]_A : A^{[t|t]} \to A^t, && f \mapsto \textstyle\bigsqcup\{f^n(\bot) \mid n \in \mathbb{N}\}
\end{aligned}$$

It can be shown that all these objects are in fact continuous.

Now, let in addition $\phi : \Omega(I) \to \underline{\text{obj}}(A)$ be a type preserving mapping, i.e., $F \in \Omega,\ \tau(F) = [w \mid i] \Rightarrow \phi(F) \in (A^w \to A^i)$. Then, $\underset{\sim}{A} := (A;\phi)$ is called a <u>(continuous) interpretation of $\Omega(I)$</u>. It implies the semantics of combinator schemes as follows.

$$[\![\ ]\!]_{\underset{\sim}{A}} : \underline{\text{rec}}(\Omega(I)) \to \underline{\text{obj}}(A)$$

is determined by

(i) $[\![S]\!]_{\underset{\sim}{A}} := [\![S]\!]_A$ for $S \in \Gamma(I)$

(ii) $[\![S]\!]_{\underset{\sim}{A}} := \phi(S)$ for $S \in \Omega(I)$

(iii) $[\![(S\, S_1 \ldots S_n)]\!]_{\underset{\sim}{A}} := [\![S]\!]_{\underset{\sim}{A}} ([\![S_1]\!]_{\underset{\sim}{A}}, \ldots, [\![S_n]\!]_{\underset{\sim}{A}})$.

Finally, we define <u>equivalence</u> of schemes $S_1, S_2 \in \underline{\text{rec}}(\Omega(I))$ as usual:

$$S_1 \sim S_2 \iff [\![S_1]\!]_{\underset{\sim}{A}} = [\![S_2]\!]_{\underset{\sim}{A}} \text{ for every interpretation } \underset{\sim}{A} \text{ of } \Omega(I).$$

6. Reduction

The language of combinator schemes is rather redundant concerning equivalence. Therefore we shall reduce this language to equivalent sublanguages. For that purpose we first select a hierarchy of types:

$$I_o := I \quad \text{and} \quad I_{n+1} := [I_n^* \mid I_n] \qquad (n \in \mathbb{N})$$

and denote by $\underline{rec}(\Omega(I))^n$ the class of combinator schemes of degree n, i.e. their type is in I_n.

(1) It can be shown that every combinator scheme is "reducible" to some $S \in \underline{rec}(\Omega(I))^n$ by means of certain abstraction and application schemes.

In the next reduction step we construct for each $\underline{rec}(\Omega(I))^n$ a subclass $\underline{\mu\text{-}rec}(\Omega(I))^n$. This construction is based on the following combinator schemes.

Lemma For all $s,t_1,\ldots,t_r,u \in \underline{typ}(I)$ and $1 \leqslant \nu \leqslant r$ the class $\underline{rec}(\Omega(I))$ contains schemes $\underline{\pi < t_1,\ldots,t_r,\nu >}$, $\underline{\sigma < s,t_1,\ldots,t_r,u >}$, $\underline{\mu < s,t_1,\ldots,t_r,u >}$, $\underline{\kappa < s,u >}$, $\underline{\alpha < s >}$ and $\underline{1}$ which have for each interpretation $\underset{\sim}{A}$ of $\Omega(I)$ the following property:

$$[\![\underline{\pi < t_1,\ldots,t_r,\nu >}]\!]_{\underset{\sim}{A}} \quad (a_1,\ldots,a_r) = a_\nu$$
$$[\![\underline{\sigma < s,t_1,\ldots,t_r,u >}]\!]_{\underset{\sim}{A}} \quad (f,f_1,\ldots,f_r) = \lambda x. \;\; f(f_1(x),\ldots,f_r(x))$$
$$[\![\underline{\mu < s,t_1,\ldots,t_r,u >}]\!]_{\underset{\sim}{A}} \quad (f,f_1,\ldots,f_r) = \lambda x. \;\; f(\mu y. \; (f_1(x,y),\ldots,f_r(x,y)))$$
$$[\![\underline{\kappa < s,u >}]\!]_{\underset{\sim}{A}} \;\; (f) = \lambda x. \; f(\,)$$
$$[\![\underline{\alpha < s >}]\!]_{\underset{\sim}{A}} \;\; (a) = \lambda(\,).\, a \quad \text{and} \quad |\underline{1}|_{\underset{\sim}{A}} = \perp \in A^e \quad .$$

The types should be clear from the context, e.g.: $\tau(\underline{\kappa < s,u >}) = [[e \mid u] \mid [s \mid u]]$.
If there is no danger of confusion type indices are dropped. In particular, we write $(\underline{\alpha}^n S)$ for n-fold abstraction.
According to their semantics these schemes are called projection schemes, substitution schemes, μ - substitution schemes, constant schemes, abstraction schemes and unit scheme.

Now, let $n \geqslant 1$. Denoting by $\Pi(I)^n$ and $\Sigma(I)^n$ the classes of projection and substitution schemes of degree n, respectively, we construct the class $B(\Omega(I))^n$ of base schemes of degree n by

$$B(\Omega(I))^1 \quad := \quad \Omega(I) \cup \Pi(I)^1$$

and

$$B(\Omega(I))^{n+1} \quad := \quad \{(\underline{\alpha}^n S) \mid S \in \Omega(I) \cup \Pi(I)^1\} \cup \Sigma(I)^{n+1} \cup \Pi(I)^{n+1} \quad .$$

Then, the class $\underline{\mu - \text{rec}}(\Omega(I))^n$ is defined as the smallest $L \subseteq \underline{\text{rec}}(\Omega(I))^n$ such that

(i) $B(\Omega(I))^n \subseteq L$

(ii) $S, S_1, \ldots, S_r \in L \Rightarrow (\underline{\sigma}\ SS_1 \ldots S_r) \in L$

(iii) $S, S_1, \ldots, S_r \in L \Rightarrow (\underline{\mu}\ SS_1 \ldots S_r) \in L$

(iv) $S \in L \Rightarrow (\underline{K}\ S) \in L$

provided that the argument schemes have suitable types. (In the terminology of WAND |7| this is called the μ - clone of $B(\Omega(I))^n$.)
From normal form theorems for μ - clones it can be derived that

(2) $\underline{\mu - \text{rec}}(\Omega(I))^1$ is equivalent to the class of regular equation schemes with parameters in the sense of GOGUEN / THATCHER |3|,

and that

(3) $\underline{\mu - \text{rec}}(\Omega(I))^2$ is equivalent to the class of recursion equation schemes with operation parameters, a slight generalization of NIVAT's recursion schemes |6|.

Moreover, combinator schemes can be reduced to these schemes:

(4) For each $n \geq 1$ and $S \in \underline{\text{rec}}(\Omega(I))^n$ there is $m \geq 1$ and $S' \in \underline{\mu - \text{rec}}(\Omega(I))^{n+m}$ such that $S \sim \underbrace{(\ldots((S'\underline{1})\underline{1})\ldots\underline{1})}_{m}$.

Since we want to use higher types only as an auxiliary device for generating low level objects our interest concentrates on the class $\underline{\text{rec}}(\Omega(I))^1$ of "operational" combinator schemes. From (4) we know that every such scheme can be obtained from some higher-type μ - clone scheme by iterated applications. This leads to the following definition:

$$\underline{\mu - \text{rec}}(\Omega(I))^n_1 := \{\underbrace{(\ldots((S\,\underline{1})\underline{1})\ldots\underline{1})}_{n-1} \mid S \in \underline{\mu - \text{rec}}(\Omega(I))^n, \ \tau(S) \text{ suitable}\}$$

Then we can prove the following hierarchy of schemes:

(5) $\underline{\mu - \text{rec}}(\Omega(I))^n_1$ is translatable into $\underline{\mu - \text{rec}}(\Omega(I))^{n+1}_1$.

Moreover, in DAMM |1| we find a proof of

(6) $\underline{\mu - \text{rec}}(\Omega(I))^2_1$ is not translatable into $\underline{\mu - \text{rec}}(\Omega(I))^1_1$.

It should be noted that (6) requires certain restrictions on Ω. E.g., if Ω contains only monadic function symbols the classes of (6) are intertranslatable.

7. Conclusion

This discussion leads to the problem whether the hierarchy of (5) is strict. A positive answer would demonstrate that procedures with recursion on higher types enlarge the computational power of a programming language.

This research is being continued in collaboration with Werner DAMM, and I take the opportunity to thank him for many helpful and stimulating discussions.

8. References

|1| DAMM, W.: Einfach-rekursive und rekursive Schemata mit stetigen Interpretationen - Diplomarbeit Bonn (1976)

|2| ENGELFRIET, J.: Simple Program Schemes and Formal Languages - Lecture Notes in Computer Science 2o (1974)

|3| GOGUEN, J.A., THATCHER, J.W.: Initial Algebra Semantics - IEEE Conf. Rec. SWAT 15 (1974), 63 - 77

|4| GREIBACH, S.A.: Theory of Program Structures: Schemes, Semantics, Verification - Lecture Notes in Computer Science 36 (1975)

|5| INDERMARK, K.: Theorie rekursiver Definitionen - Vorlesung, Universität Bonn (1975)

|6| NIVAT, M.: On the Interpretation of Recursive Program Schemes - IRIA - Rapport de Recherche 84 (1974)

|7| WAND, M.: A Concrete Approach to Abstract Recursive Definitions - in: Automata, Languages, and Programming (ed. NIVAT), Amsterdam (1973), 331 - 341