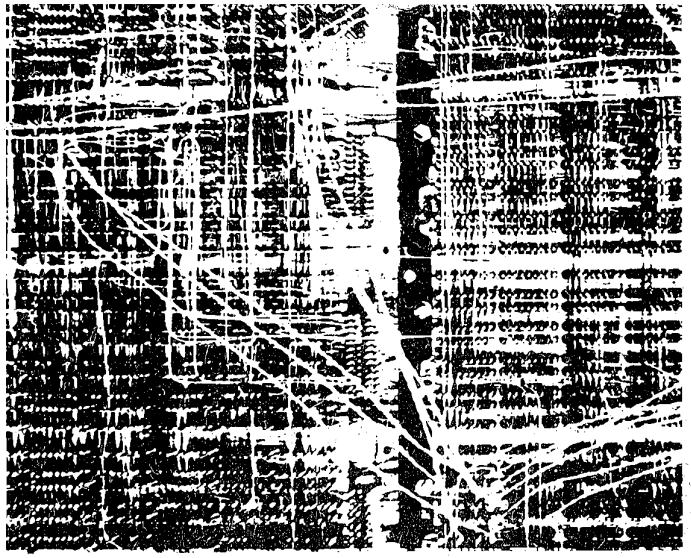


ACTAS DE LA **PRIMERA CONFERENCIA INTERNACIONAL EN CIENCIA DE LA COMPUTACION**

Auspician:
Banco de Chile •
Secico •
A.F.P. Santa María •
Leniz y Silva, •
Ingenieros Consultores
Cientec •
Cecinas Winter •
Revista Informativa •

PONTIFICIA UNIVERSIDAD
CATOLICA DE CHILE

UNIVERSIDAD DE CHILE

PRIMERA CONFERENCIA INTERNACIONAL

EN CIENCIA DE LA COMPUTACION

FECHA : 24 - 27 AGOSTO

LUGAR : CASA CENTRAL
PONTIFICIA UNIVERSIDAD
CATOLICA DE CHILE

Especificación Metódica de Tipos Abstractos de Datos a Través de Reglas de Reescritura

Especificação Metódica de Tipos de Dados Abstratos Através de Regras de Reescrita

Methodical Specification of Abstract Data Types Via Rewrite Rules

Paulo A. S. Veloso
Departamento de Informática
Pontificia Universidad Católica
do Rio de Janeiro
Brasil

INTRODUCTION

Abstraction has been widely recognized as a powerful programming tool [Liskov '75, Guttag '77] and several representation-independent approaches to its semantical specification have been proposed [Liskov , Zilles '75]. In particular, abstract data types (ADT's) specified by (conditional) equations have been frequently employed [Goguen et al. '77 ; Guttag et al. '78; Gaudel '79].

But, whoever has tried to provide a formal specification for a model has probably faced some difficulties in this error-prone task [Kapur '79]. The problems amount basically to

(i) what axioms to write ?
(ii) are they correct ?
(iii) are they enough ?

Here a methodology to alleviate these difficulties is presented. It is based on the concepts of canonical term algebra [Goguen et al. '77] and of term-rewriting systems [Huet, Oppen '80].

The presentation will include an example, which is simple enough to allow the detailed application of the method. The latter will also be justified in general and its wide applicability will be apparent.

MODEL

The problem of specification can be posed as follows: given a model find, if possible, a specification (in a fixed formalism) for it. In the initial algebra approach, we are given as model a (many-sorted) algebra M, every element of which is reachable, i.e. denoted by a variable-free term. We want a (finite) set of (conditional) equations E such that M is (isomorphic to) the initial algebra of the class of similar algebras satisfying E [Goguen et al. '77].

As our running example we consider the simple case of strings of elements from some non-empty set, assumed specified (cf. Fig. 0).

We have an informal description of the model M to be specified. It frequently happens that one receives informal - sometimes even vague or ambiguous - descriptions to start with.

As a first step towards a more precise presentation of the model, we can give a formal description of the syntax of the language: sorts and operation symbols. We employ underlined words for the syntax (cf. Fig. 1).

Now we can give a precise and unambiguous description of the model M, generally couched in a mathematical notation, which may introduce extraneous concepts in order to achieve the goal of formalizing the presentation. We shall use the corresponding non-underlined italic words for the denotations in M of the syntactic symbols

Model (informal)

Sorts

. Alph : some given non-empty set of letters ;

. Bool : the set of logical values : +, - ;

. Str : finite sequences of elements of Alph.

Operations

. true = +

. false = -

. same : tests equality of letters;

. if-then-else - : conditional;

. null = empty sequence, of length zero ;

. make : creates a unit-length sequence out of a letter;

. cons : adds a letter in front of a sequence;

. append : concatenation of sequences ;

. equal : tests equality of sequences.

Figure 0

Syntax (formal)

Sorts : Str , Alph , Bool .

Operations

	→	Bool	:	true ; false
(Bool, Bool, Bool)	→	Bool	:	if-then-else-
(Alph, Alph)	→	Bool	:	same
	→	Str	:	null
(Alph)	→	Str	:	make
(Alph, Str)	→	Str	:	cons
(Str , Str)	→	Str	:	append
(Str , Str)	→	Bool	:	equal

Figure 1

(cf. Fig. 2).

It often happens that an informal presentation gives a better intuitive feeling of the model. Accordingly, it may be used as a starting point in the "creative" part of obtaining a formal specification. But of course, one cannot prove the equivalence of a formal description with an informal one. It is mainly for the purpose of checking the correctness and completeness of the formal especifications obtained that we use a formal description of the model.

Model (formal)

Domains :

- *Alph* = A (some given non-empty set)
- *Bool* = { -, + } (with - ≠ +)
- *Str* = A^* = $\{< a_1,\ldots,a_n > / a_1,\ldots,a_n \in A;\ n \in \mathbb{N}\}$

Operations :

- *true* = +
- *false* = -
- *if* α *then* β *else* $\gamma = \begin{cases} \beta & \text{if } \alpha = + \\ \gamma & \text{if } \alpha = - \end{cases}$
- *same* (a,b) $= \begin{cases} + & \text{if } a = b \\ - & \text{if } a \neq b \end{cases}$
- *null* = < >
- *make*(a) = <a>
- *cons* (a, $<a_1,\ldots,a_m>$) = $<a, a_1,\ldots,a_m>$
- *append* ($< a_1,\ldots,a_m>$, $<b_1,\ldots,b_n>$) =

  $= < a_1,\ldots,a_m, b_1,\ldots,b_n>$
- *equal* ($<a_1,\ldots,a_m>$ $<b_1,\ldots,b_n>$) $= \begin{cases} + & \text{if } m = n \text{ and } a_i = b_i,\ i = 1,\ldots,n \\ - & \text{otherwise} \end{cases}$

Figure 2

CANONICAL TERM MODEL

A major step towards axiomatically specifying the model (described formally or not) consists of replacing it by another one with syntactic domains and precisely defined operations on these syntactic objects.

Here, the crucial part is the choice of a canoni - cal form for the elements of M. For, even a short sequence as <a,b> can be obtained in various ways

$cons(a,make(b))$, $append(make(a), make(b))$,...,

among which we choose one [$cons(a,cons(b,null))$] to represent <a,b>

Letting T denote the set of all variable-free terms of the language and T be the corresponding term algebra (Herbrand algebra) the situation is a follows. As T is initial in the class of all algebras similar to it, giving an algebra M amounts to giving a (unique and surjective) homomorphism $h: T \to M$ so that $M \cong T/\equiv[h]$.
We want a set $R \subseteq T$ (i.e. $R_s \subseteq T_s$ for each sort $s \in S$) such that

(a) the restricition g of h to R is a bijection;
(b) for each operation symbol $\underline{f}$ and terms $t_1,\ldots,t_n \in T$, whenever the term $\underline{f}\ t_1,\ldots,t_n$ is in R then so are $t_1,\ldots,t_n$.

Notice that condition (a) says that each element of M has a unique term in R representing it. In fact, this amounts to selecting a uniform method for constructing all

elements of the model (cf. Fig. 3).

Now we use the bijection $g: R \to M$ to induce operations on R, as follows.

(c) for each operation symbol $\underline{f}$ and all terms $r_1,\ldots,r_n \in R$, set

$$F(r_1,\ldots,r_n) = \bar{g}^{1}(\underline{f}[g(r_1),\ldots,g(r_n)])$$

As a result we obtain an algebra $R \tilde{=} M$ (cf. Fig. 4).

Notice that R obtained above to satisfy (a), (b) and (c) is indeed a canonical term algebra (cta) in the sense of [Goguen et al. '77] as it is easily seen that whenever $\underline{f}\ t_1,\ldots t_n \in R$ then $F(t_1,\ldots,t_n) = \underline{f}\ t_1,\ldots,t_n$.

Now as R and M are isomorphic, it can be more convenient and reliable to work with R instead of M, in order to exploit the syntactical nature of the elements of R. We may regard R as a special subset of T, the elements of which have their meaning specified (via g). Our task then consists of relating the rest of T to R. Also, notice that R is <u>not</u> a subalgebra of T. Indeed, our next step will consist of, so to speak, providing ways to bring back into R the results of operations that have escaped from it.

Canonical Terms

$R_{\underline{Alph}} = C \subseteq A$ (assumed given)

$R_{\underline{Bool}} = \{ \underline{true}, \underline{false} \}$

$R_{\underline{Str}} = \{ \underline{cons}(c_n,\ldots,\underline{cons}(c_j,\ldots,\underline{cons}(c_1,\underline{null})\ldots)\ldots) \;/\; c_1,\ldots, c_j,\ldots, c_n \in C,\ n \in \mathbb{N} \}$

[Convention : case n = 0 is null]

Alternative (recursive) definition of $R_{\underline{Str}}$:

$R_{\underline{Str}}$ is the least subset of $T_{\underline{Str}}$ such that

- $\underline{null} \in R_{\underline{Str}}$
- whenever $c \in C$ and $r \in R_{\underline{Str}}$ then $\underline{cons}(c,r) \in R_{\underline{Str}}$

Figure 3

Canonical Term Algebra R

Domains :

- ALPH = $R_{\underline{Alph}}$ = C
- BOOL = $R_{\underline{Bool}}$
- STR = $R_{\underline{Str}}$

Operations :

T : TRUE = true

F : FALSE = false

$$\text{I : IF } \lambda \text{ THEN } \mu \text{ ELSE } \nu = \begin{cases} \mu & \text{if } \lambda = \underline{\text{true}} \\ \nu & \text{if } \lambda = \underline{\text{false}} \end{cases}$$

$$\text{S : SAME [c,d]} = \begin{cases} \underline{\text{true}} & \text{if } c = d \\ \underline{\text{false}} & \text{if } c \neq d \end{cases}$$

N : NULL = null

M : MAKE [c] = cons (c,null)

$$\text{C : CONS}[c,\underline{cons}(c_m,\ldots,\underline{cons}(c_1,\text{null})\ldots)] = \underline{cons}(c,\underline{cons}(c_m,\ldots,\underline{cons}(c_1,\text{null})\ldots))$$

$$\text{A : APPEND}[\underline{cons}(c_m,\ldots,\underline{cons}(c_1,\text{null})\ldots), \underline{cons}(d_n,\ldots,\underline{cons}(d_1,\text{null})\ldots)] = \underline{cons}(c_m,\ldots,\underline{cons}(c_1,\underline{cons}(d_n,\ldots,\underline{cons}(d_1,\text{null})\ldots))\ldots)$$

$$\text{E : EQUAL}[\underline{cons}(c_m,\ldots,\underline{cons}(c_1,\text{null})\ldots),\underline{cons}(d_n,\ldots,\underline{cons}(d_1,\text{null})\ldots)] = \begin{cases} \underline{\text{true}} & \text{if } m = n \text{ and } c_j = d_j \text{ for } j = 1,\ldots,n \\ \underline{\text{false}} & \text{otherwise} \end{cases}$$

Figure 4

TRANSFORMATION RULES

We now search for a (finite) set Γ of term-rewriting rules with the following properties

(I) Completeness: for each operation symbols $\underline{f}$ and all $r_1,\ldots,r_n \in R, \underline{f}\ r_1,\ldots r_n \xrightarrow{*} F(r_1,\ldots,r_n)$

(II) Consistency: whenever $t \xrightarrow{*} t'$ then $h(t) = h(t')$.

Completeness of Γ means that its rules are powerful enough to, according to (I), transform the result of operating on canonical terms into the corresponding values in the cta R (cf. Fig. 5).

The problem we face now is the creation of such rules. Here, the very explicit (and often recursive) nature of the canonical form helps in suggesting a strategy to achieve this goal (frequently by reducing it to simpler subgoals).

It we follow the method, carefully checking each step, we obtain a system Γ of term-rewriting rules, which is both complete and consistent (cf. Fig. 6).

Consistency of Γ means that its rules are sound on M (or R), in that for each rule $t(\vec{v}) \to t'(\vec{v})$ of Γ we have the sentence $\forall\ \vec{v}\ [t(\vec{v}) = t'(\vec{v})]$ satisfied in R. This gives a good method to check the consistency of Γ, namely checking whether for all $\vec{r}$ in R $T(\vec{r}) = T'(\vec{r})$, where T and T', respectively are the denotations of t and t' in the cta R.

The importance of checking the rules should not be overlooked. Firstly, this is what guarantees the goals of completeness and consistency. Secondly, each rule can be

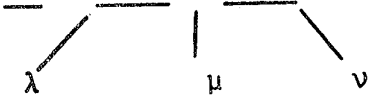
Transformations

(P) $a \xrightarrow{*} c$ (were $c \in C$, and $h(c) = h(a)$)

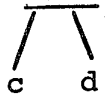
(T) $\underline{\text{true}} \xrightarrow{*} \underline{\text{true}}$

(F) $\underline{\text{false}} \xrightarrow{*} \text{false}$

(I) $\underline{\text{if}}$ - $\underline{\text{then}}$ - $\underline{\text{else}}$ - $\xrightarrow{*}$ $\begin{cases} \mu & \text{if } \lambda = \underline{\text{true}} \\ \nu & \text{if } \lambda = \underline{\text{false}} \end{cases}$

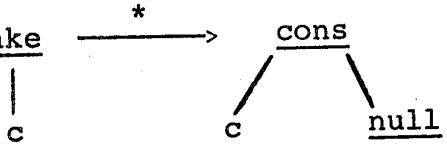


(S) $\underline{\text{same}} \xrightarrow{*} \begin{cases} \underline{\text{true}} & \text{if } c = d \\ \underline{\text{false}} & \text{if } c \neq d \end{cases}$

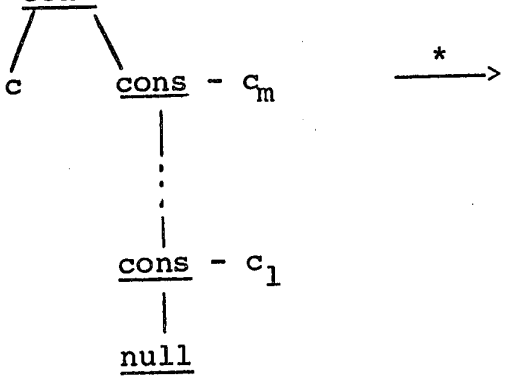


(N) $\underline{\text{null}} \xrightarrow{*} \underline{\text{null}}$

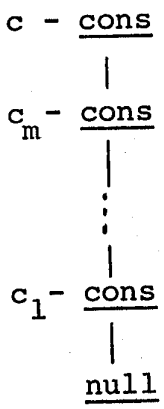
(M) $\underline{\text{make}} \xrightarrow{*} \underline{\text{cons}}$



(C) $\underline{\text{cons}}$



$\xrightarrow{*}$



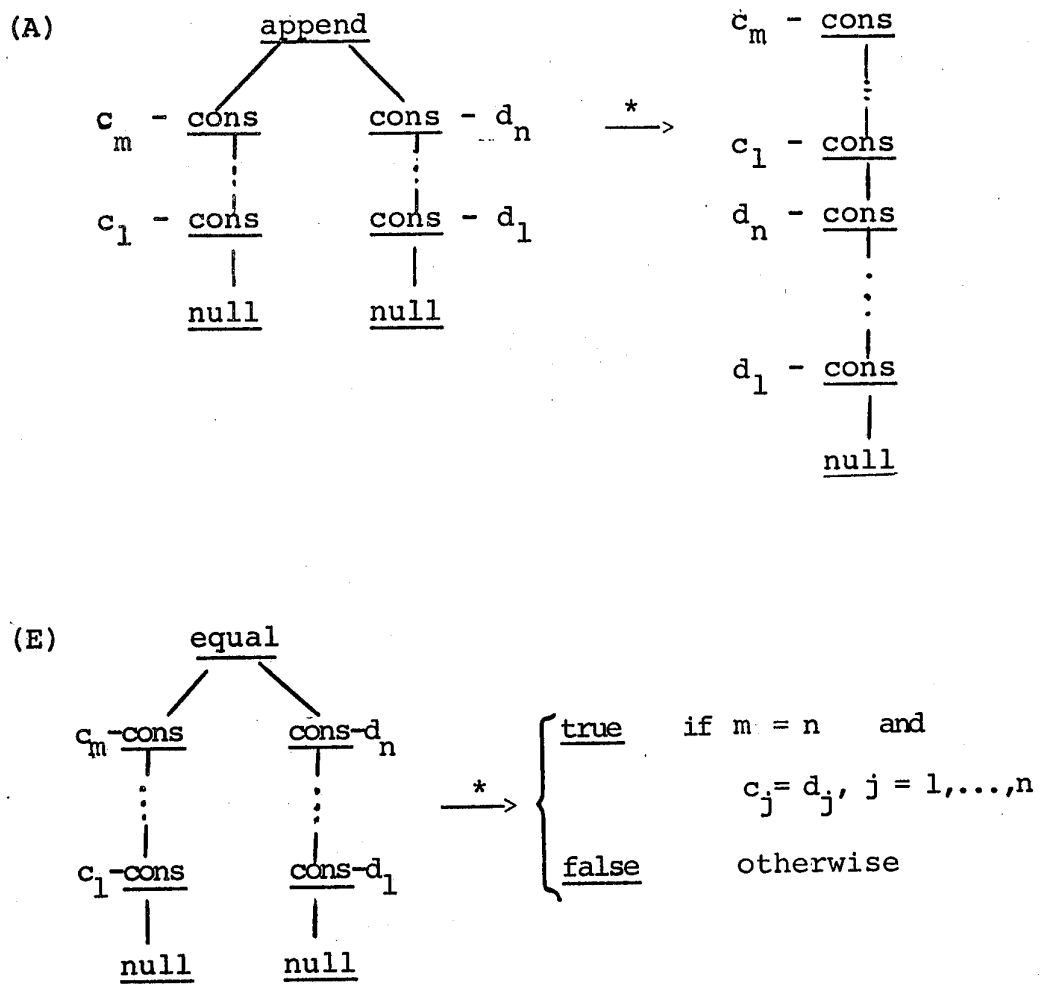

(A)
append
c_m - cons
cons - d_n
c_1 - cons
cons - d_1
null
null
*
c_m - cons
c_1 - cons
d_n - cons
d_1 - cons
null
(E)
equal
c_m-cons
cons-d_n
c_1-cons
cons-d_1
null
null
*
true
false
if m = n and
$c_j = d_j$, j = 1,...,n
otherwise

Figure 5

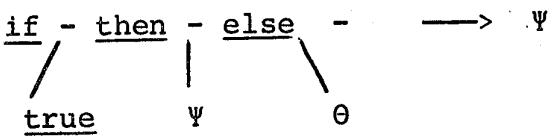
tested separately for consistency, whereas for completeness it is a set of rules that has to be checked powerful enough to achieve a transformation, Thirdly, when some candidate-rules fail to be sound or to achieve a desired transformation generally the very test helps pinpointing the trouble-spots and suggesting appropriate corrections.

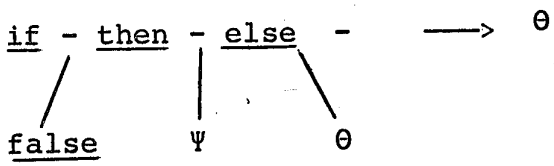
Rules

(p) assumed given

(t) none necessary

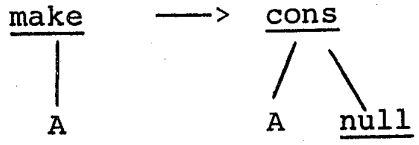
(f) none necessary

(it) 

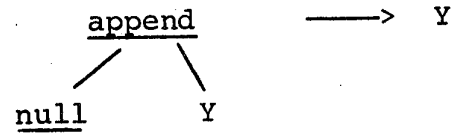
(if) 

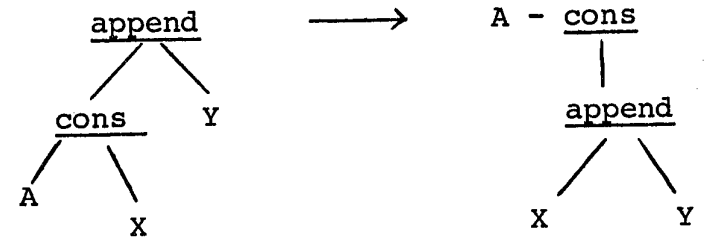
(s) assumed given

(n) none necessary

(m) 

(c) none necessary

(an) 

(ac) 

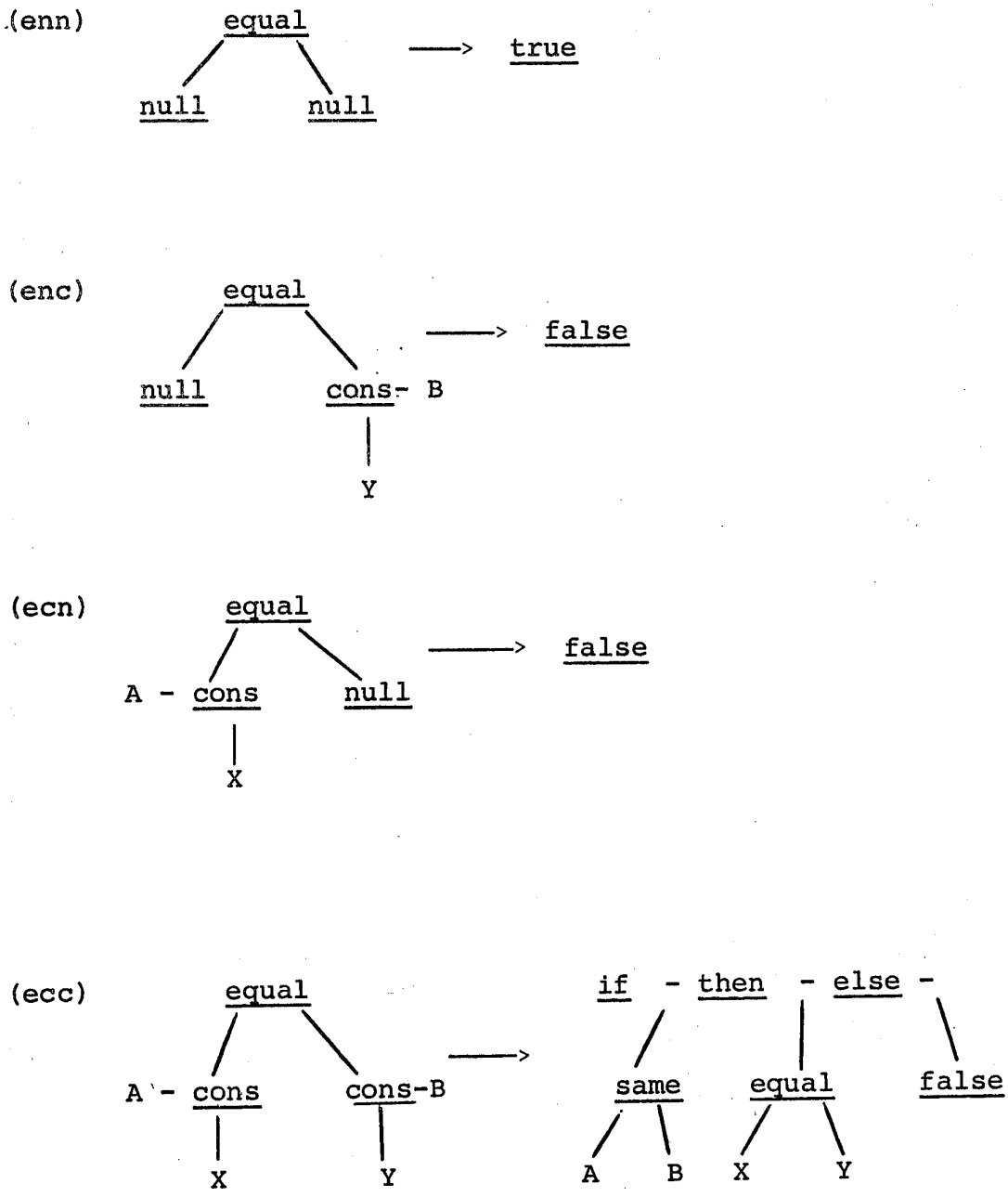

(enn)
equal
null
null
true
(enc)
equal
null
cons- B
Y
false
(ecn)
equal
A - cons
null
X
false
(ecc)
equal
A - cons
cons-B
X
Y
if - then - else -
same
equal
false
A
B
X
Y

Figure 6

EQUATIONS

Having checked the rewriting system Γ to be consistent and complete, in the sense of (I) and (II) of the preceding section, we already have a formal specifi - cation for M. But, each rule $t \to t'$ of Γ corresponds naturally to an equation $t=t'$. Thus it is straightforward to transform Γ into a set Δ of equations (cf. Fig. 7).

This set Δ of equations obtained from Γ clearly has the following properties

(1) for each operation symbol $\underline{f}$ and all $r_1,\ldots,r_n \in R$

$\underline{f}\ r_1,\ldots,r_n \in R$, $\underline{f}\ r_1,\ldots r_n \equiv F(r_1,\ldots,r_n)$ $[\Delta]$

(2) R satisfies every equation of Δ

(Here $\equiv [\Delta]$ denotes the congruence on T generated by Δ)

These conditions are sufficient (and necessary) for the cta R to be isomorphic to $T/\equiv[\Delta]$, according to theorem 9 of [Goguen et al '77]. Hence, Δ is a (correct and complete) algebraic specification for M, in that $M \cong T/\equiv[\Delta]$.

It should be clear that the above straightforward obtention of Δ from Γ is possible due to the context-free nature of the rules. If a rule of Γ has some context conditions then one has to try simulating its effects by means of formulas more complex than equations.

Equations

(π) assumed given

($\iota\tau$) if true then ψ, else $\theta = \psi$

($\iota\phi$) if false then ψ else $\theta = \theta$

(σ) assumed given

(μ) make(a) = cons(a,null)

($\alpha\nu$) append(null,y) = y

($\alpha\kappa$) append(cons(a,x),y)= cons(a, append(x,y))

($\varepsilon\nu\nu$) equal(null,null) = true

($\varepsilon\nu\kappa$) equal(null,cons(b,y)) = false

($\varepsilon\kappa\upsilon$) equal(cons(a,x), null) = false

($\varepsilon\kappa\kappa$) equal(cons(a,x), cons(b,y)) = if same(a,b)
then equal(x,y)
else false

Figure 7

CONCLUDING REMARKS

The methodology proposed here for the algebraic specification of an ADT, given by means of a (formal or informal) model M, proceeds via the following steps

. canonical term algebra R;

. term rewriting system Γ;

. set of equations Δ.

Another way to describe the methodology is by regarding it as the process of obtaining various inter - mediate "consistent and complementary specifications" [Donahue '76 ; Levy '77] for M, each one with its own distinctive features.

The cta R has the advantage of being a precisely described model with well-structured syntactic domains, the operations of which can be defined without resorting to formal variables ranging over the sorts being specified.

The rewriting system Γ can be viewed as a "generative-transformational specification". Indeed by constructing Γ to be a finitely terminating Church - Rosser system [Huet, Oppen '80], R consists exactly of the irreducible elements of Γ and each $t \epsilon T$ reduces to a unique $r \epsilon R$. In addition, by making the application of the rules of Γ deterministic, we easily obtain a "procedural specifica - tion", which amounts to an abstract implementation of the ADT on the type of terms. Such procedural specifications [Furtado, Veloso '81; Furtado, Veloso, Castilho '81] can be simply translated into executable programs in a symbol-manipulating language, thereby providing the

opportunity for early usage and experimentation, without the need to resort to a specially designed system as in [Gannon et al '80; Guttag et al '78; Goguen et al, Jan '77] Another application appears in [Remy, Veloso '81].

This methodology is apparently very helpful in guiding the search for a formal specification. Of course the crucial step is the election of a "good" canonical form. Even though an initial cta is known to exist [Goguen et al '77], the very form of its terms heavily influences the form of the resulting specification and how easy it is to obtain it. In this choice a good intuitive understanding of the model plays an important role. Besides, a convenient description of the canonical form may have to employ concepts (such as lexicographical ordering of terms) not in the language.

Finally, it may be worth mentioning that the methodology presented is <u>not</u> intended to decide whether a finite (or effective, etc.) specification exists or not. It just goes ahead trying to obtain one.

REFERENCES

. Donahue, J.E. - Complementary definitions of programming language semantics. Springer-Verlag, 1976

. Furtado, A.L.; Veloso, P.A.S. - "Procedural specifications and implementations for abstract data types". SIGPLAN NOTICES, to appear, 1981.

. Furtado, A.L.; Veloso, P.A.S. ; Castilho, J.M.V. de - "Verification and testing of simple entity-relationship representations". PUC/RJ, Dept. Informática. Res. Rept., Apr. 1981.

. Gannon, J.; McMullin, P.; Hamlet, R.; Ardis, M. - "Testing traversable stacks". SIGPLAN NOTICES, vol. 15 (nº 1), Jan. 1980.

. Gaudel, M.C. - "Algebraic specification of abstract data types". IRIA, Res. Rept. 360, 1979.

. Goguen, J.; Tardo, J.; Williamson, N.; Zamfir, M. - "A practical method for testing algebraic specifications". The UCLA Comp. Sci. Dept. Quarterly, vol. 7 (nº 1), Jan. 1977

. Goguen, J.A.; Thatcher, J. W. ; Wagner, E.G. - "An initial algebra approach to the specification, correctness and implementation of abstract data types" in R. T. Yeh (ed.) Current treends in programming methodology, vol IV, Prentice-Hall, 1977.

. Guessarian, I. - "Algebraic semantics". Res. Rept. 80-13, L.I.T.P., Paris, Mar. 1980

. Guttag, J.V.; Horowitz, E.; Musser, D.R. - "Abstract data types and software validation". Comm. ACM, vol. 21 (nº 12), Dec. 1978

. Huet, G.; Oppen, D.C. - "Equations and rewrite rules: a survey". Stanford Univ., Comp. Sci. Dept. STAN-CS-80-785, 1980.

. Kapur, D. - "Specifications of Majster's traversable stack and Veloso's traversable stack". SIGPLAN NOTICES , vol. 14 (nº5), May 1979

. Levy, M.R. - "Some remarks on abstract data types" SIGPLAN NOTICES, vol. 12 (nº 7), Jul. 1977

. Liskov, B. H. - "Data types and program correctness" SIGPLAN NOTICES, July 1975

. Liskov, B.; Zilles, S. - "Specification techniques for data abstractions" . IEEE Trans. Software Engin,,vol.SE-1 (nº 1), 1975

. Remy, J.L.; Veloso, P.A.S. - "Comparing abstract data type specifications via their normal forms". PUC/RJ, Dept. Informática, MCC nº 1/81, March, 1981.