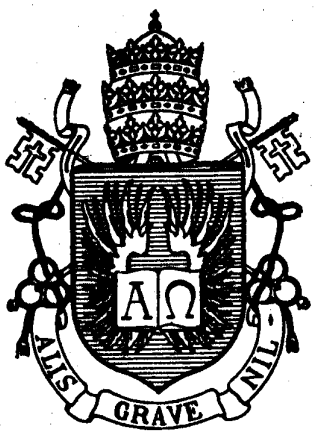


PUC

Series: Monografias em Ciência da Computação

Nº 7/81

METHODICAL SPECIFICATION OF ABSTRACT DATA TYPES VIA REWRITING SYSTEMS

by

P. A. S. Veloso

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 – CEP-22453
RIO DE JANEIRO – BRASIL

Series: Monografias em Ciência da Computação

Nº 7/81

Series Editor: Marco A. Casanova June 1981

METHODICAL SPECIFICATION OF ABSTRACT DATA TYPES VIA REWRITING SYSTEMS *

by

P.A.S. Veloso

* Research partly sponsored by FINEP, CNPq and the French Ministry for Foreign Affairs

ABSTRACT

A data type is often given by an informal model. Its formal specification is an important task, but also difficult and error-prone. Here a methodology for this task is presented. Its steps are, first, the election of a canonical form defining a canonical term algebra; second, a system of sound rewriting rules powerful enough to achieve the syntactical transformations of the canonical term algebra. The final translation of rewriting rules into equations is immediate. The methodology is illustrated by the detailed presentation of a simple example.

Key works

abstract data types, formal specification, canonical terms, rewriting systems, specification methodology.

RESUMO

Um tipo de dados costuma ser apresentado através de um modelo informal, sendo sua especificação formal uma tarefa importante, porêm difícil e sujeita a erros. Aqui se apresenta uma metodologia para sistematizar essa tarefa. Suas etapas são, primeiro, escolha de uma forma canônica, o que define uma álgebra de termos canônicos; segundo, um sistema de regras de reescrita consistentes que seja poderoso o suficiente para efetuar as transformações sintáticas da álgebra de termos canônicos. A tradução final de regras de reescrita em equações é imediata. Ilustra-se a metodologia pela apresentação detalhada de sua aplicação a um exemplo simples.

Palavras Chaves

tipos abstratos de dados, especificação formal, termos canônicos, sistemas de reescrita, metodologia para especificação.

ACKNOWLEDGEMENTS

Part of the research report herein took place at the Laboratoire d' Informatique Théorique et Programmation, U. E. R. de Mathématiques, Université de Paris VII, in October 1980. Helpful discussions with Irène Guessarian (LITP) and Jean-Luc Remy (CRIN, Nancy) were quite influential. Partial financial support from the Brazilian National Council for Scientific and Technological Development and the French Ministry for Foreign Affairs is gratefully acknowledged.

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

together with

some set of operations generating a set D of variable-free terms of sort Alph.

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^* = $\{\langle a_1,\ldots,a_n\rangle \;/\; a_1,\ldots,a_n \in A;\ n \in \mathbb{N}\}$

Operations :

- *true* = +
- *false* = −
- *if* α *then* β *else* γ = $\begin{cases}\beta & \text{if } \alpha = +\\ \gamma & \text{if } \alpha = -\end{cases}$
- *same* (a,b) = $\begin{cases}+ & \text{if } a = b\\ - & \text{if } a \neq b\end{cases}$
- *null* = $\langle\ \rangle$
- *make*(a) = $\langle a\rangle$
- *cons* (a, $\langle a_1,\ldots,a_m\rangle$) = $\langle a, a_1,\ldots,a_m\rangle$
- *append* ($\langle a_1,\ldots,a_m\rangle$, $\langle b_1,\ldots,b_n\rangle$) = $= \langle a_1,\ldots,a_m, b_1,\ldots,b_n\rangle$
- *equal* ($\langle a_1,\ldots,a_m\rangle$ $\langle b_1,\ldots,b_n\rangle$) = $\begin{cases}+ & \text{if } m = n \text{ and } a_i = b_i,\ i = 1,\ldots,n\\ - & \text{otherwise}\end{cases}$

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 canonical 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 (cf. Fig. 3)

(a) the restriction 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

Canonical Terms

$R_{\underline{Alph}} = C \subseteq D$ such that $h: T \to M$ establishes a one-to-one correspondence from C onto A.

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

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

[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

elements of the model (cf. Appendix 1).

Now we use the bijection $g: R \to M$ to induce operations on R, as follows (cf. Appendix 2)

(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 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)

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

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

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

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 symbol $\underline{f}$ 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 ; cf. Appendix 3).

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 $t(\vec{v}) \rightarrow t'(\vec{v})$ 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 , (cf. Appendix 4).

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) $d \xrightarrow{*} c'$ (whenever $c \in C$, $d \in D$, and $h(c) = h(d)$)

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

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

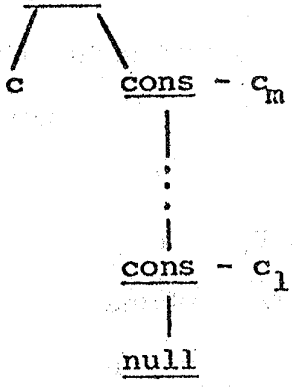
(I) $\underline{\text{if}}\ -\ \underline{\text{then}}\ -\ \underline{\text{else}}\ - \xrightarrow{*}$ $\begin{cases} \mu & \text{if } \lambda = \underline{\text{true}} \\ \nu & \text{if } \lambda = \underline{\text{false}} \end{cases}$

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

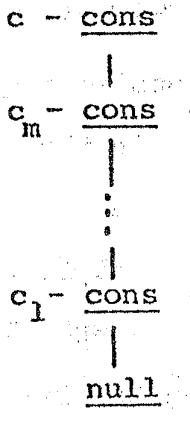
(N) $\underline{\text{null}} \xrightarrow{*} \underline{\text{null}}$

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

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



$\xrightarrow{*}$



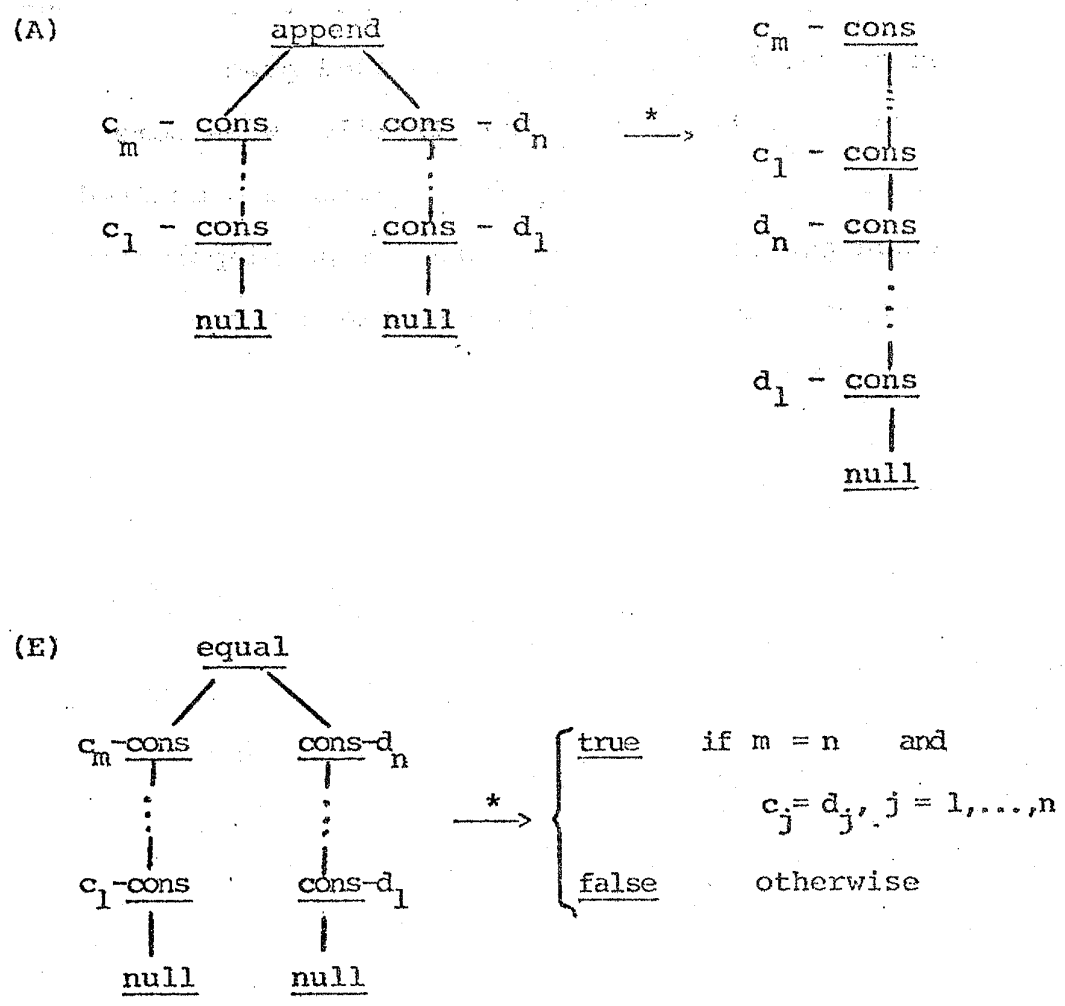

(A)
append
c_m - cons
cons - d_n
c_1 - cons
cons - d_1
null
null
*
c_m - cons
c_1 - cons
d_n - cons
d_1 - cons
null
(E)
equal
c_m-cons
cons-d_n
c_1-cons
cons-d_1
null
null
*
true
if m = n and
$c_j = d_j$, j = 1,...,n
false
otherwise

Figure 5

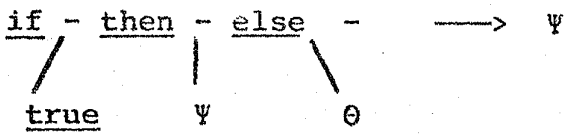
tested separately for consistency, whereas for completeness it is a set of rules that has to be checked powerful enough to achieve a transformation. Thirdly, when some candidate-rules fail to be sound or to achieve a desired transformation generally the very test helps pinpointing the trouble-spots and suggesting appropriate corrections. (Appendix 5 contains the proof of completeness for our running example.)

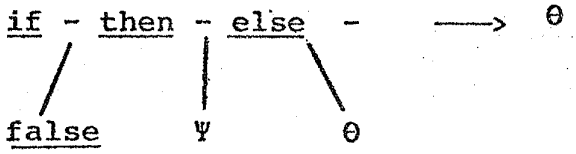
Rules

(p) assumed given

(t) none necessary

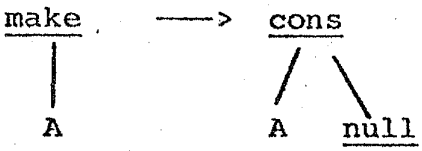
(f) none necessary

(it) 

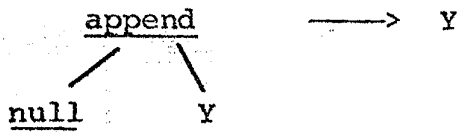
(if) 

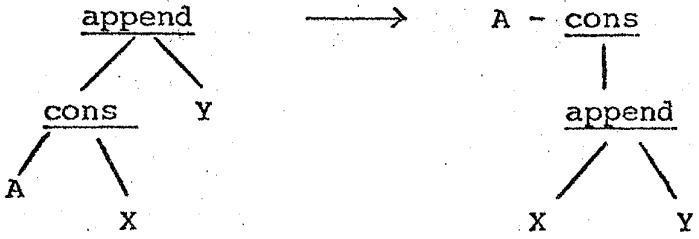
(s) assumed given

(n) none necessary

(m) 

(c) none necessary

(an) 

(ac) 

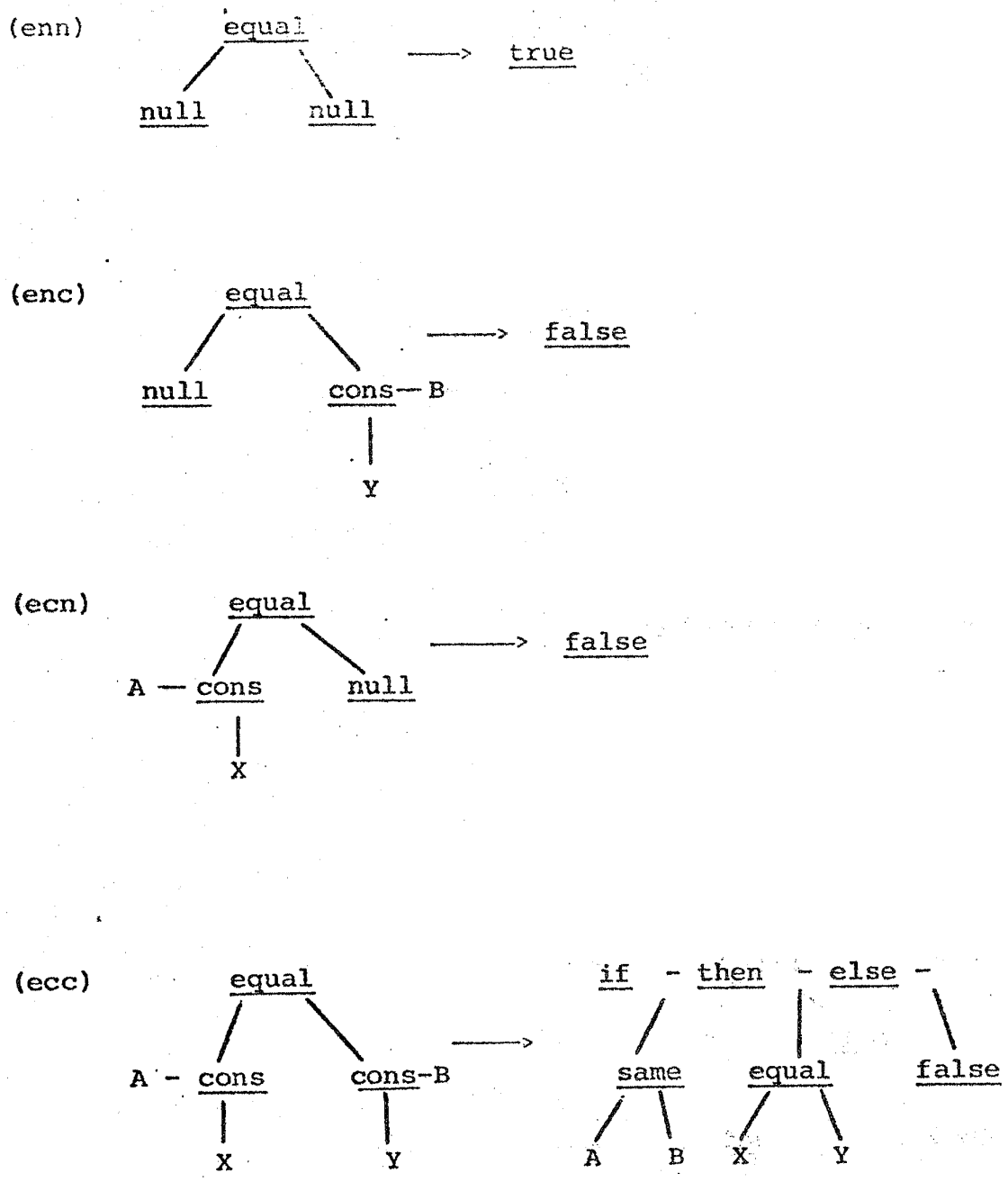

(enn)
equal
null
null
true
(enc)
equal
null
cons
B
Y
false
(ecn)
equal
A
cons
null
X
false
(ecc)
equal
A
cons
cons-B
X
Y
if
then
else
same
equal
false
A
B
X
Y

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 natural ly 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 translation of Γ into Δ 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

(ιτ) if true then ψ else θ = ψ

(ιφ) if false then ψ else θ = θ

(σ) assumed given

(μ) make(a) = cons(a,null)

(αν) append(null,y) = y

(ακ) append(cons(a,x),y)= cons(a, append(x,y))

(ενν) equal(null,null) = true

(ενκ) equal(null,cons(b,y)) = false

(εκυ) equal(cons(a,x), null) = false

(εκκ) 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 $t \in T$ reduces to a unique $r \in R$. 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.

It should also be apparent that the methodology is applicable to parametric data types. In fact, only slight modifications are required to view our example as the parametric data type strings-of - [ ].

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, vol. 16 (nº 3), Mar. 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. MCC nº 4/81, Apr. 1981 (also 2nd. Int. Conf. Entity-Relationship Appr.)

. 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.

APPENDIX 1

Here we check that R indeed satisfies the conditions (a) and (b) required of the canonical representatives.

First, the restriction g of h to R is

$\cdot g_{\underline{Alph}} : C \to Alph = A$

$c \mapsto c$ (assumed given)

$\cdot g_{\underline{Bool}} : R_{\underline{Bool}} \to Bool = \{+,-\}$

$g_{\underline{Bool}}[\underline{true}] = h_{\underline{Bool}}[\underline{true}] = true = +$

$g_{\underline{Bool}}[\underline{false}] = h_{\underline{Bool}}[\underline{false}] = false = -$

$\cdot g_{\underline{Str}} : R_{\underline{Str}} \to Str = A^*$

$$g_{\underline{Str}}[\underline{cons}(c_n,\ldots,\underline{cons}(c_j,\ldots,\underline{cons}(c_1,\underline{null})\ldots)\ldots)] =$$

$$= h_{\underline{Str}}[\underline{cons}(c_n,\ldots,\underline{cons}(c_j,\ldots,\underline{cons}(c_1,\underline{null})\ldots)\ldots)] =$$

$$= cons(h_{\underline{Alph}}(c_n),\ldots,cons(h_{\underline{Alph}}(c_j),\ldots,cons(h_{\underline{Alph}}(c_1),$$

$$h_{\underline{Str}}(\underline{null}))\ldots)\ldots) =$$

$$= cons(c_n,\ldots,cons(c_j,\ldots,cons(c_1,<>)\ldots)\ldots) =$$

$$= <c_n,\ldots,c_j,\ldots,c_1>$$

(a) For each sort s, g_s is a bijection

- $\underline{Alph}$: $g_{\underline{Alph}}$ is assumed bijective.
- $\underline{Bool}$: $g_{\underline{Bool}}$ is clearly a bijection.
- $\underline{Str}$: We check first surjectiveness then injectiveness.

(i) $g_{\underline{Str}}$ is surjective

Take an arbitrary element of $Str = A^*$, say $< a_1,\ldots,a_n >$ with $a_1,\ldots a_n \in A$ and $n \in \mathbb{N}$. As $g_{\underline{Alph}}$ is assumed surjective we have $c_1,\ldots,c_n \in C$ with $g_{\underline{Alph}}(c_j) = h_{\underline{Alph}}(c_j) = a_j$ for $j=1,\ldots,n$. Now take $r = \underline{cons}(c_1,\ldots,\underline{cons}(c_n,\underline{null})\ldots)$. The definition of g gives $g_{\underline{Str}}[r] = < a_1,\ldots,a_n >$.

(ii) $g_{\underline{Str}}$ is injective

Let $c_1,\ldots,c_m$, $d_1,\ldots,d_n \in C$ and assume

$g_{\underline{Str}}[\underline{cons}(c_1,\ldots,\underline{cons}(c_m,\underline{null})\ldots)] =$

$= g_{\underline{Str}}[\underline{cons}(d_1,\ldots,\underline{cons}(d_n,\underline{null})\ldots)]$.

The definition of g then gives $<c_1,\ldots c_m> = <d_1,\ldots,d_n>$.

Whence $m=n$ and for $j=1,\ldots,m$ $c_j=d_j$.

Thus, for each $j=1,\ldots,m$ $g_{\underline{Alph}}(c_j) = c_j =$

$= d_j = g_{\underline{Alph}}(d_j)$, and the assumed injectiveness of $g_{\underline{Alph}}$ yields $c_j=d_j$. Therefore $\underline{cons}(c_1,\ldots,\underline{cons}(c_m,\underline{null})\ldots) =$

$= \underline{cons}(d_1,\ldots,\underline{cons}(d_n,\underline{null})\ldots)$.

(b) Whenever $\underline{f}t_1\ldots t_n$ is in R then so are $t_1,\ldots,t_n$.

. $\underline{Alph}$: assumed.

. $\underline{Bool}$: clear.

. $\underline{Str}$: If $\underline{f}t_1\ldots t_n \in R_{\underline{Str}}$ then

either $\underline{f} = \underline{null}$ and $n=0$;

or $\underline{f}$ = $\underline{cons}$ and n=2 .

In the latter case, we have

$\underline{f}\ t_1\ \ t_2 = \underline{cons}(c_m, \underline{cons}(c_{m-1}, \ldots, \underline{cons}(c_1, \underline{null})\ldots))$

for some $m > 0$ and $c_1, \ldots, c_{m-1}, c_m \in C$.

Then $t_1 = c_m \in C = R_{\underline{Alph}}$

and $t_2 = \underline{cons}(c_{m-1}, \ldots, \underline{cons}(c_1, \underline{null})\ldots) \in R_{\underline{Str}}$.

Notice that condition (b) is purely syntactical and all that was used in checking it was the definition of R. On the other hand, condition (a) refers to the model M and some of its properties were used in checking condition (a).

APPENDIX 2

Operations induced on R by g

. TRUE $= g^{-1}_{\underline{Bool}}[true] = g^{-1}_{\underline{Bool}}[+] = \underline{true}$

. FALSE $= g^{-1}_{\underline{Bool}}[false] = g^{-1}_{\underline{Bool}}[-] = \underline{false}$

. IF λ THEN μ ELSE $\upsilon = g^{-1}_{\underline{Bool}}(if\ g_{\underline{Bool}}[\lambda]\ then\ g_{\underline{Bool}}[\mu]\ else\ g_{\underline{Bool}}[\upsilon]) =$

$$= g^{-1}_{\underline{Bool}}\left(\begin{cases} g_{\underline{Bool}}[\mu] & \text{if} \quad g_{\underline{Bool}}[\lambda] = + \\ g_{\underline{Bool}}[\upsilon] & \text{if} \quad g_{\underline{Bool}}[\lambda] = - \end{cases}\right)$$

$$= \begin{cases} \mu & \text{if} \quad \lambda = \underline{true} \\ \upsilon & \text{if} \quad \lambda = \underline{false} \end{cases}$$

. SAME [c,d] $= g^{-1}_{\underline{Bool}}[same(g_{\underline{Alph}}[c], g_{\underline{Alph}}[d])] =$

$$= g^{-1}_{\underline{Bool}}\left(\begin{cases} + & \text{if} \quad g_{\underline{Alph}}[c] = g_{\underline{Alph}}[d] \\ - & \text{if} \quad g_{\underline{Alph}}[c] \neq g_{\underline{Alph}}[d] \end{cases}\right)$$

$$= \begin{cases} \underline{true} & \text{if} \quad c = d \\ \underline{false} & \text{if} \quad c \neq d \end{cases}$$

. MAKE [c] = $g^{-1}_{\underline{Str}}$ [*make*($g_{\underline{Alph}}$[c])] =

= $g^{-1}_{\underline{Str}}$[*make*(c)] = $g^{-1}_{\underline{Str}}$[<c>] = $\underline{cons}$(c,$\underline{null}$)

. CONS[c,$\underline{cons}$(c_m,...,$\underline{cons}$(c_1,$\underline{null}$)...)] =

= $g^{-1}_{\underline{Str}}$[*cons*($g_{\underline{Alph}}$[c],$g_{\underline{Str}}$[$\underline{cons}$(c_m,...,$\underline{cons}$(c_1,$\underline{null}$)...)])]=

= $g^{-1}_{\underline{Str}}$[*cons*(c, <c_m,...c_1>)] = $g^{-1}_{\underline{Str}}$[< c,c_m,...,c_1>] =

= $\underline{cons}$(c,$\underline{cons}$(c_m,...,$\underline{cons}$(c_1,$\underline{null}$)...))

. APPEND [$\underline{cons}$(c_m,...,$\underline{cons}$(c_1,$\underline{null}$)...)],

$\underline{cons}$(d_n,...,$\underline{cons}$(d_1,$\underline{null}$)...)] =

= $g^{-1}_{\underline{Str}}$ [*append*($g_{\underline{Str}}$[$\underline{cons}$(c_m,...,$\underline{cons}$(c_1,$\underline{null}$)...)] ,

$g_{\underline{Str}}$[$\underline{cons}$(d_n,...,$\underline{cons}$(d_1,$\underline{null}$)...)])] =

= $g^{-1}_{\underline{Str}}$ [*append*(<c_m,...,c_1> , <d_n,...,d_1>)] =

= $g^{-1}_{\underline{Str}}$[<c_m,...,c_1,d_n,...d_1>] =

= $\underline{cons}$(c_m,...,$\underline{cons}$(c_1,$\underline{cons}$(d_n,...,$\underline{cons}$(d_1,$\underline{null}$)...))...)

APPENDIX 3

Strategies and subgoals for the transformations

P) Assumed given.

T) None necessary.

F) None necessary.

I) Decompose into two cases: $\lambda =$ true and $\lambda =$ false .

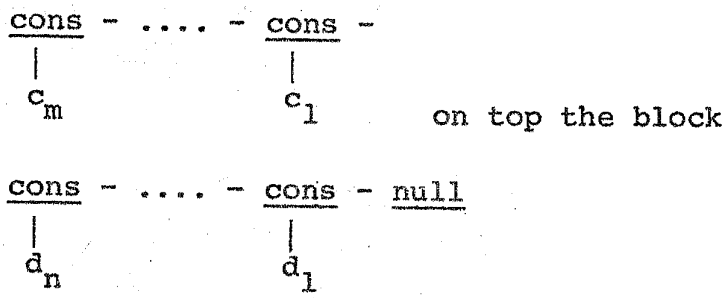
S) Assumed given.

N) None necessary.

M) Immediate, as the transformation itself can be made into a rule.

C) None necessary .

A) The transformation calls for putting the block



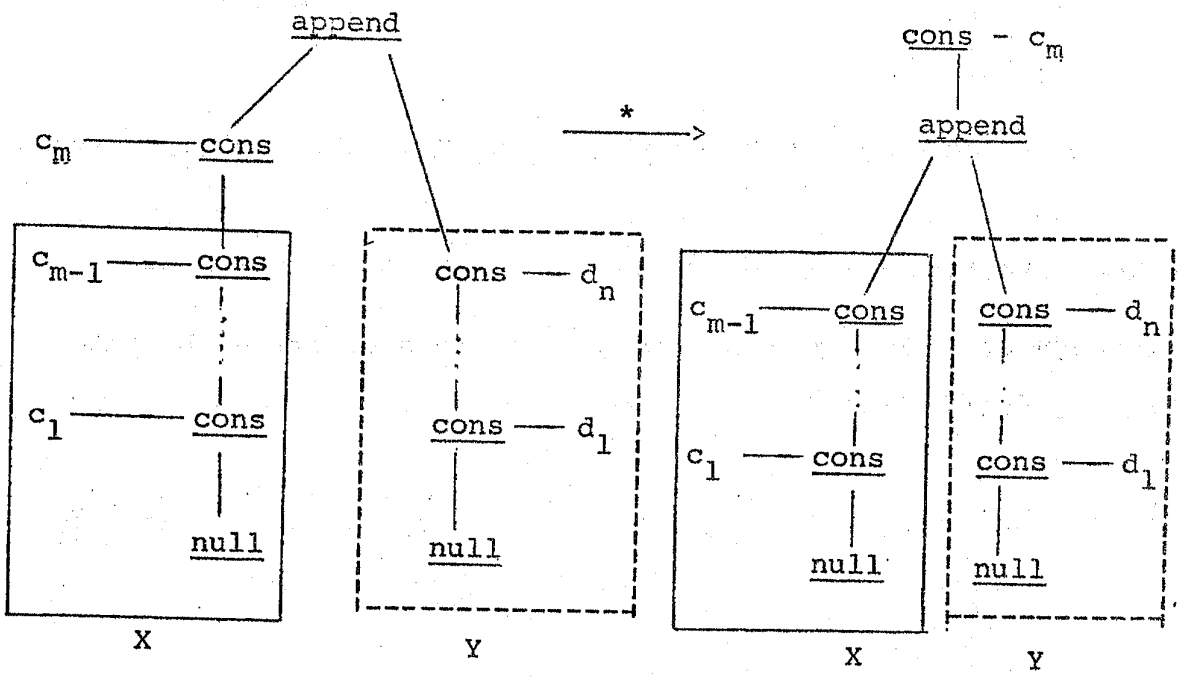
In other words, we are required to move all the c_i's (respecting their relative order) to the top the right subtree.

A possible strategy for this consists of

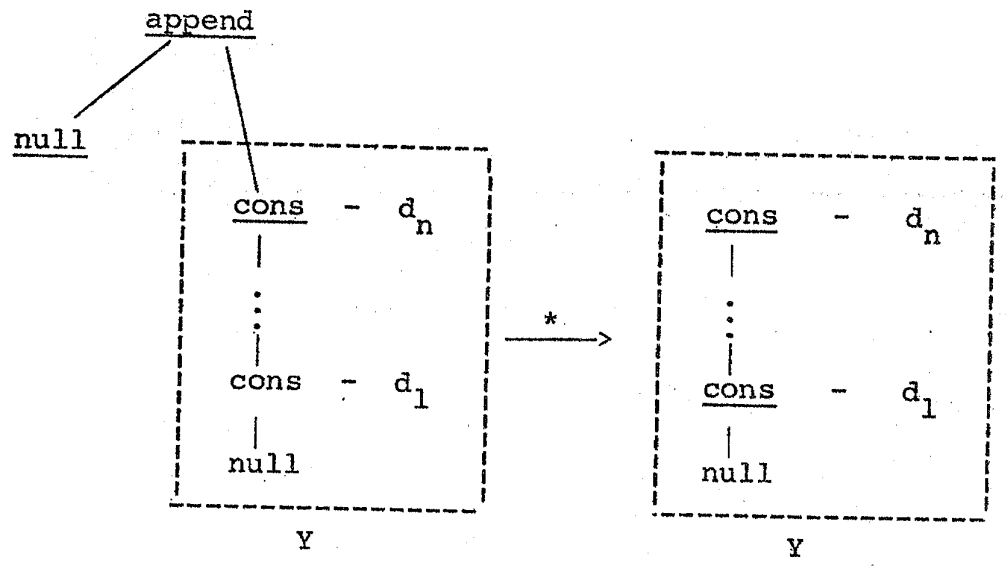
- first, repeatedly moving the top c_i to the top of the right subtree;
- then, treating the case of no c_i's remaining in the left subtree.

This suggests decomposing the transformation (A) into two as follows

A1)



A2)

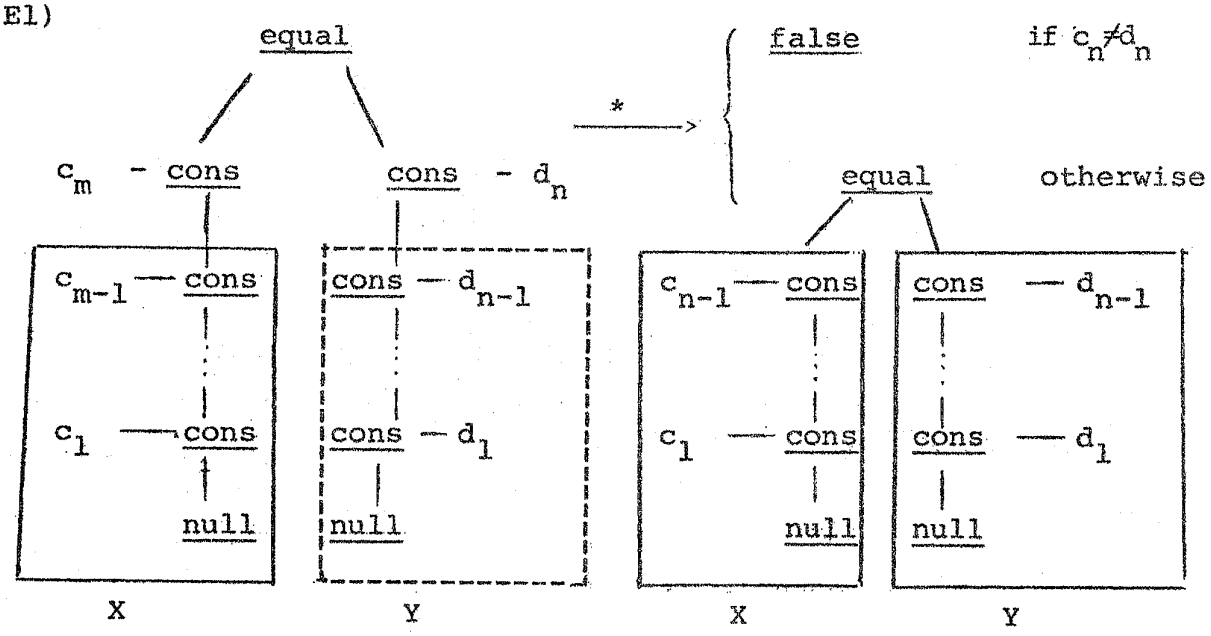


E) The transformation is to give <u>true</u> if the left and right subtrees are identical, and <u>false</u> otherwise. In other words, we are required to compare both subtrees. A possible way of achieving this consists of

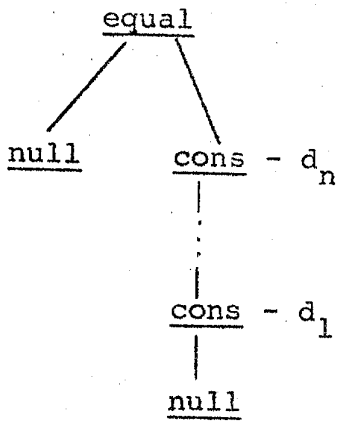
- first, repeatedly comparing the top c_i with the top d_j, deleting both;
- then, treating the cases of no c_i's or no d_j's remaining.

This suggests the decomposition of the transformation (E) into three as follows

E1)

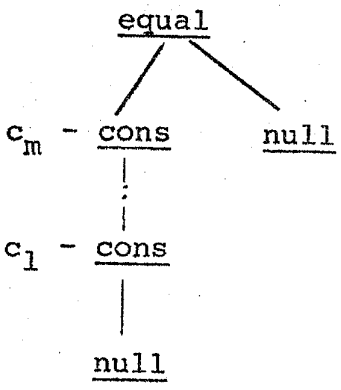


E2)



$$\xrightarrow{*} \begin{cases} \underline{\text{false}} & \text{if } n \neq o \\ \underline{\text{true}} & \text{if } n = o \end{cases}$$

E3)



$$\xrightarrow{*} \begin{cases} \underline{\text{true}} & \text{if } m = 0 \\ \underline{\text{false}} & \text{if } m = 0 \end{cases}$$

Now, E2 subdividies into cases $n \neq 0$ and $n = 0$ and E3 splits into cases $m = 0$ and $m \neq 0$. Since $n = 0$ in E2 coincides with $n = 0$ in E3, we have 3 cases after all, all of them simple.

APPENDIX 4

Consistency of the set Γ of rules

(it) For every $\mu, \upsilon \in$ BOOL we have, by the definition of the operation IF_THEN_ELSE on R (cf. Fig. 4)

IF $\underline{\text{true}}$ THEN μ ELSE $\upsilon = \mu$

(if) Similarly, for $\mu, \upsilon \in$ BOOL

IF $\underline{\text{false}}$ THEN μ ELSE $\upsilon = \upsilon$

(m) For every $c \in$ ALPH, we have, by the definition in Fig. 4

$$\text{MAKE}[c] \overset{(\underline{M})}{=} \underline{\text{cons}}(c, \underline{\text{null}}) \quad \text{and}$$

$$\text{CONS}[c, \text{NULL}] \overset{(\underline{N})}{=} \text{CONS}[c, \underline{\text{null}}] \overset{(\underline{C})}{=} \underline{\text{cons}}(c, \underline{\text{null}})$$

(an) For every $s \in$ STR, s is of the form $s =$

$= \underline{\text{cons}}(d_n, \ldots, \underline{\text{cons}}(d_1, \underline{\text{null}})\ldots)$, with $d_1, \ldots d_n \in C$ (cf. Fig. 3). Thus

APPEND[NULL,s] = (by N)

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

$= \underline{\text{cons}}(d_n, \ldots, \underline{\text{cons}}(d_1, \underline{\text{null}})\ldots) = s$

(ac) For every $r, s \in$ STR, we have, say

$r = \underline{\text{cons}}(c_m, \ldots, \underline{\text{cons}}(c_1, \underline{\text{null}})\ldots)$

$s = \underline{\text{cons}}(d_n, \ldots, \underline{\text{cons}}(d_1, \underline{\text{null}})\ldots)$

So, for every $c \in$ ALPH, the lefthand side gives

APPEND [CONS [c,r],s] =

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

$= \text{APPEND}\ [\underline{\text{cons}}\ (c, \underline{\text{cons}}(c_m, \ldots \underline{\text{cons}}(c_1, \underline{\text{null}})\ldots)),$

$\underline{\text{cons}}\ (d_n, \ldots, \underline{\text{cons}}(d_1, \underline{\text{null}})\ldots)] =$ (by A)

$= \underline{\text{cons}}(c, \underline{\text{cons}}(c_m, \ldots \underline{\text{cons}}(c_1, \underline{\text{cons}}(d_n, \ldots, \underline{\text{cons}}(d_1, \underline{\text{null}})\ldots))\ldots))$

Now, the right hand side gives

CONS[c, APPEND[r,s]] =

= CONS[c, APPEND [$\underline{cons}(c_m,\ldots,\underline{cons}(c_1,\underline{null})\ldots)$,

$\underline{cons}(d_n,\ldots,\underline{cons}(d_1,\underline{null})\ldots)$] = (by A)

= CONS[c, $\underline{cons}(c_m,\ldots,\underline{cons}(c_1,\underline{cons}(d_n,\ldots,\underline{cons}(d_1,\underline{null})\ldots))\ldots)$] (by C.)

= $\underline{cons}(c,\underline{cons}(d_n,\ldots,\underline{cons}(c_1,\underline{cons}(d_n,\ldots,\underline{cons}(d_1,\underline{null})\ldots))\ldots))$

Thus, both sides agree.

(enn) Clearly

lhs = EQUAL[NULL,NULL] $\overset{(N)}{=}$ EQUAL[$\underline{null}$,$\underline{null}$] $\overset{(E)}{=}$ $\underline{true}$

rhs = TRUE $\overset{(T)}{=}$ $\underline{true}$

(enc) For every $d \in$ ALPH and every s =

= $\underline{cons}(d_n,\ldots,\underline{cons}(d_1,\underline{null})\ldots) \in$ STR, we have

lhs = EQUAL [NULL, CONS[d,$\underline{cons}(d_n,\ldots,\underline{cons}(d_1,\underline{null})\ldots)$]]=

= EQUAL[$\underline{null}$,$\underline{cons}(d,\underline{cons}(d_n,\ldots,\underline{cons}(d_1,null)\ldots))$] =

= $\underline{false}$

rhs = FALSE = $\underline{false}$

(ecn) Similarly to (enc), we have

EQUAL [CONS[c,r], NULL] = $\underline{false}$ = FALSE

(ecc) Given c, $d \in$ ALPH and r, $s \in$ STR, say

r = $\underline{cons}(c_m,\ldots,\underline{cons}(c_1,\underline{null})\ldots)$

s = $\underline{cons}(d_n,\ldots,\underline{cons}(d_1,\underline{null})\ldots)$

we have

$$\text{lhs} = \text{EQUAL}[\text{CONS}[c,\underline{\text{cons}}(c_m,\ldots,\underline{\text{cons}}(c_1,\underline{\text{null}})\ldots)]\ ,$$
$$\text{CONS}[d,\underline{\text{cons}}(d_n,\ldots,\underline{\text{cons}}(d_1,\underline{\text{null}})\ldots)]] =$$

$$(\underline{\underline{C}})\quad \text{EQUAL}[\underline{\text{cons}}(c,\underline{\text{cons}}(c_n,\ldots,\underline{\text{cons}}(c_1,\underline{\text{null}})\ldots)),$$
$$\underline{\text{cons}}(d,\underline{\text{cons}}(d_n,\ldots,\underline{\text{cons}}(d_1,\underline{\text{null}})\ldots))]$$

$$(\underline{\underline{E}})\begin{cases}\underline{\text{true}} & \text{if } m = n \text{ and } c_1 = d_1,\ldots,c_m = d_n,\ c = d\\ \underline{\text{false}} & \text{otherwise}\end{cases}$$

On the other hand

rhs = IF SAME[c,d]THEN EQUAL[r,s] ELSE FALSE =

$$(\underline{\underline{S}})\begin{cases}\text{IF } \underline{\text{true}} \text{ THEN EQUAL}[r,s] \text{ ELSE FALSE} & \text{if } c = d\\ \text{IF } \underline{\text{false}} \text{ THEN EQUAL}[r,s] \text{ ELSE FALSE} & \text{if } c \neq d\end{cases}$$

$$(\underline{\underline{I}})\begin{cases}\text{EQUAL}[\underline{\text{cons}}(c_m,\ldots,\underline{\text{cons}}(c_1,\underline{\text{null}})\ldots) & \\ \qquad \underline{\text{cons}}(d_n,\ldots,\underline{\text{cons}}(d_1,\underline{\text{null}})\ldots)] & \text{if } c = d\\ \text{FALSE} & \text{if } c \neq d\end{cases}$$

$$(\underline{\underline{E}})\begin{cases}\begin{cases}\underline{\text{true}} & \text{if } m = n \text{ and } c_j = d_j\ (j=1,\ldots n)\\ \underline{\text{false}} & \text{otherwise}\end{cases} & \text{if } c = d\\ \text{FALSE} & \text{if } c \neq d\end{cases}$$

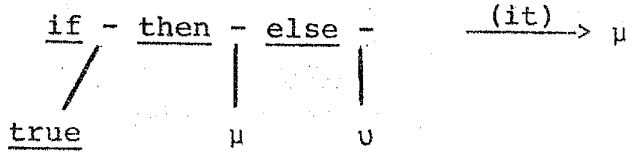
$$(\underline{\underline{F}})\begin{cases}\underline{\text{true}} & \text{if } m = n \text{ and } c_1 = d_1,\ldots,c_m = d_m,\ c=d\\ \underline{\text{false}} & \text{otherwise}\end{cases}$$

APPENDIX 5

Completeness of the rewriting system Γ

We have to check that Γ is powerful enough to achieve the transformations (I), (M), (A) and (E), since (P) and (S) are assumed achieved and the other transformations are trivial.

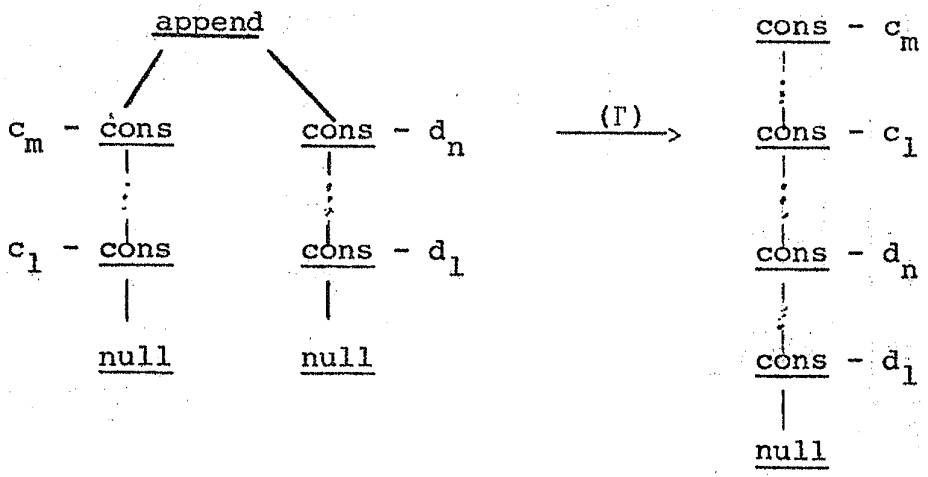
(I) For λ = true, we have



and for λ = false , (if) will do the job.

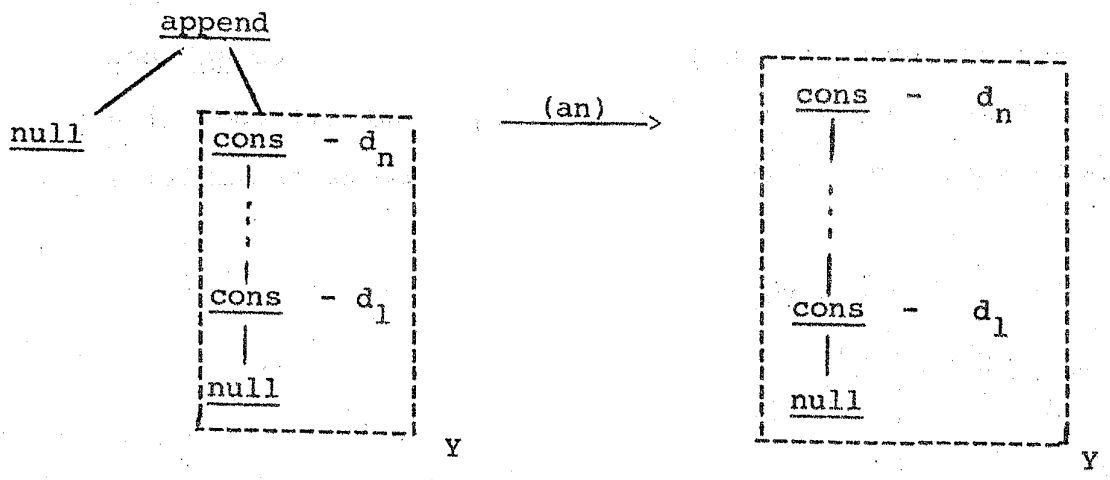
(M) Clear, by (m) .

(A) We will show

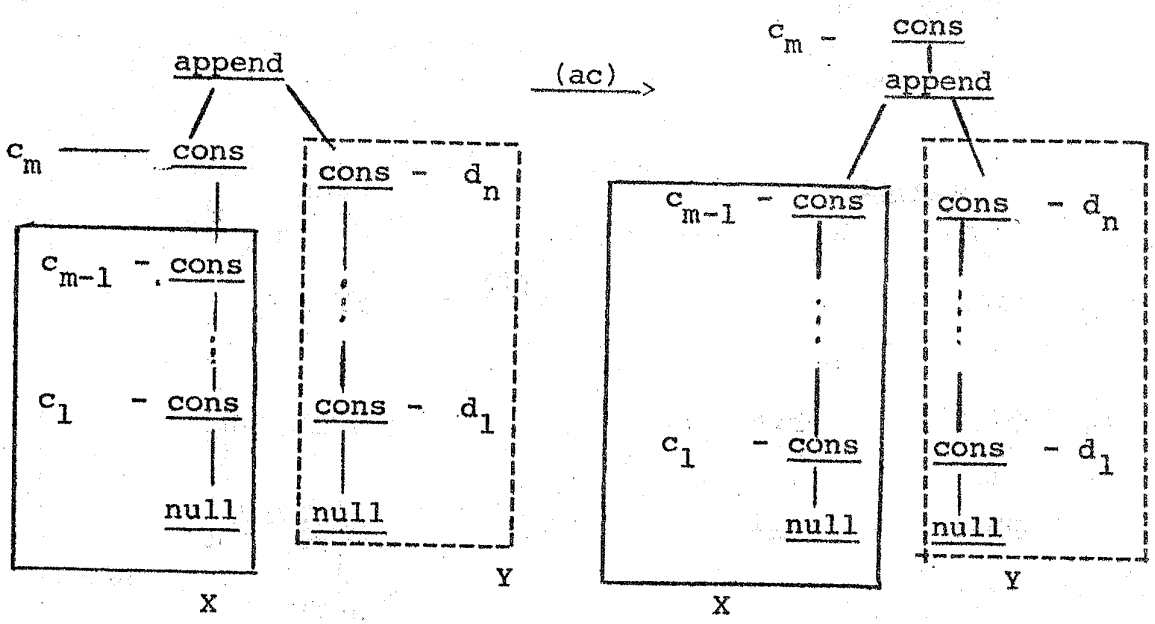


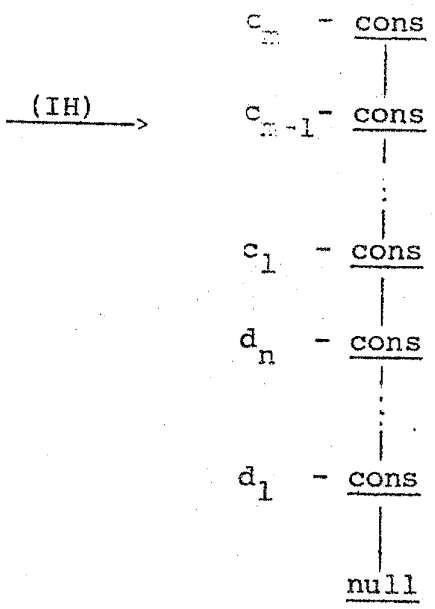
by induction on : m.

Case m = 0

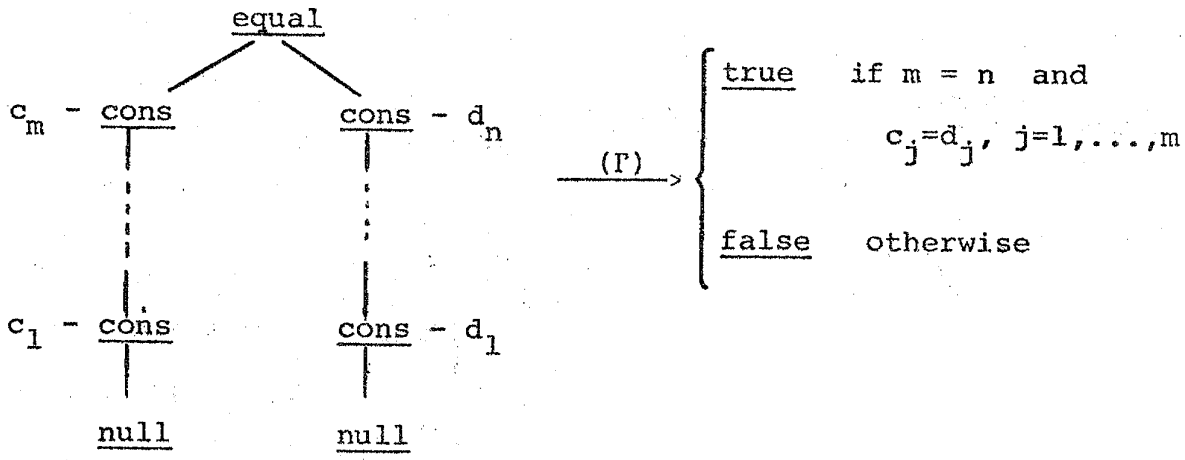


Case m > 0; assuming, as IH, the result for m-1



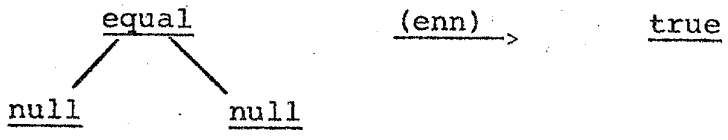


(E) We will show, by induction on m,

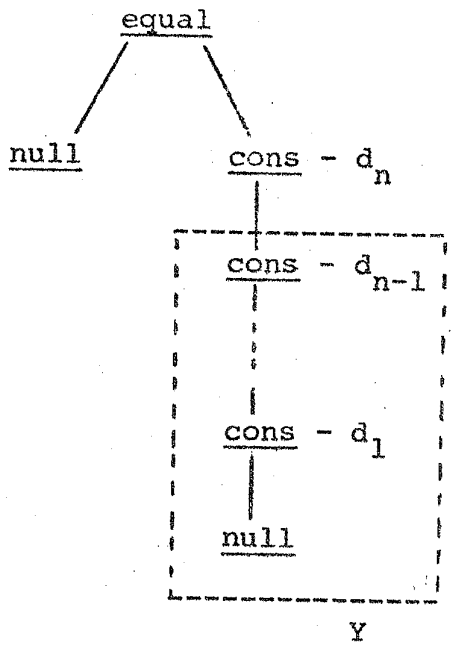


. Case m = 0

- subcase n = 0. Here m = 0 = n and

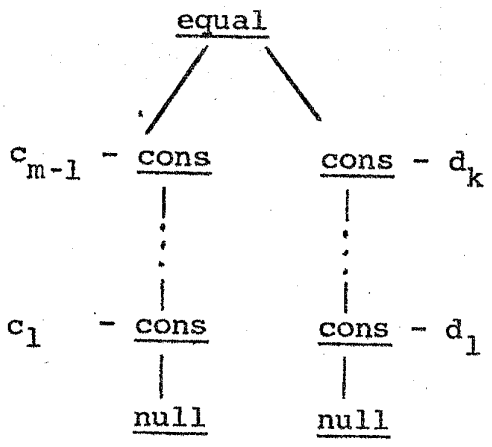


- subcase $n > 0$. Here $m = 0 \neq n$ and



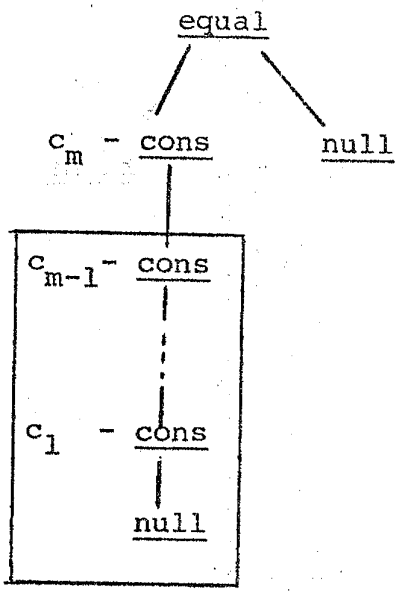
$\xrightarrow{(enc)}$ false

. Case $m > 0$; assuming, as (IH) : for each k



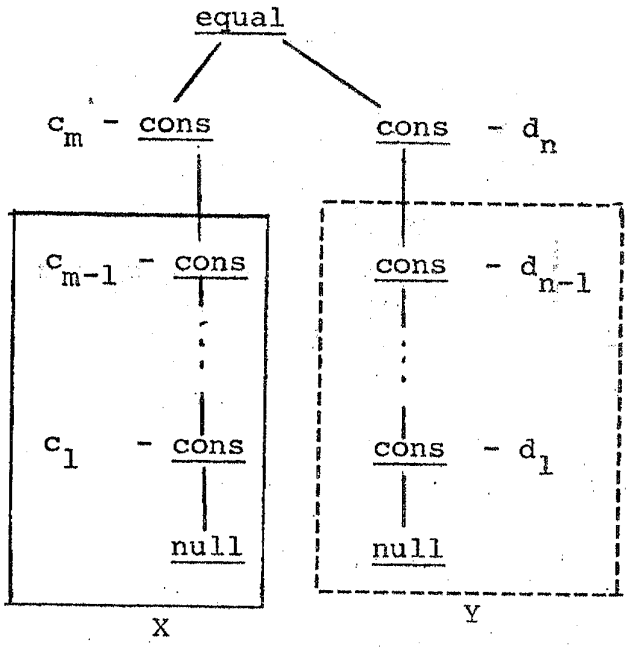
$\xrightarrow{(\Gamma)}$ $\begin{cases} \text{true} & \text{if } m-1 = k \text{ and} \\ & c_j = d_j,\ j=1,\ldots,m-1 \\ \text{false} & \text{otherwise} \end{cases}$

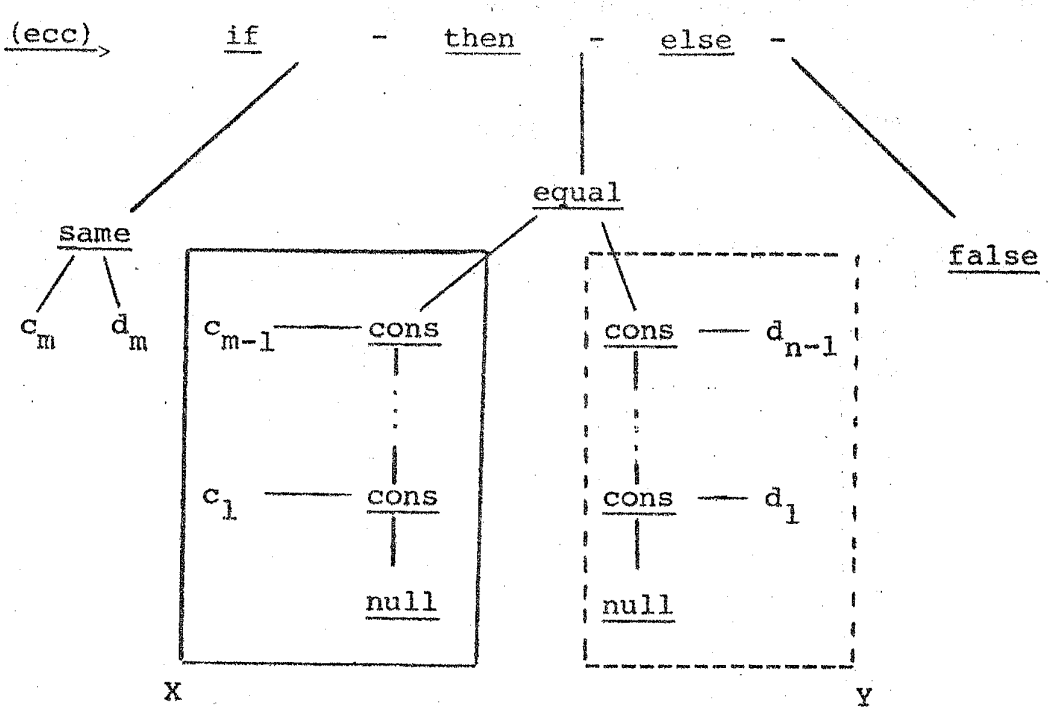
- subcase $n = 0$. Here $m \neq 0 = n$ and

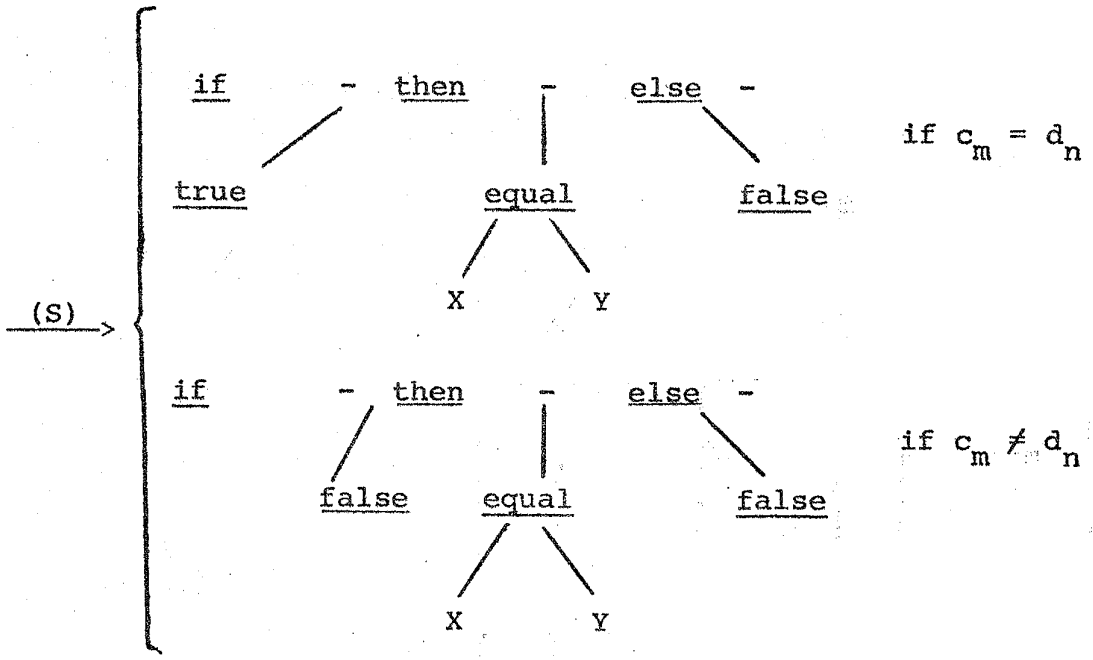


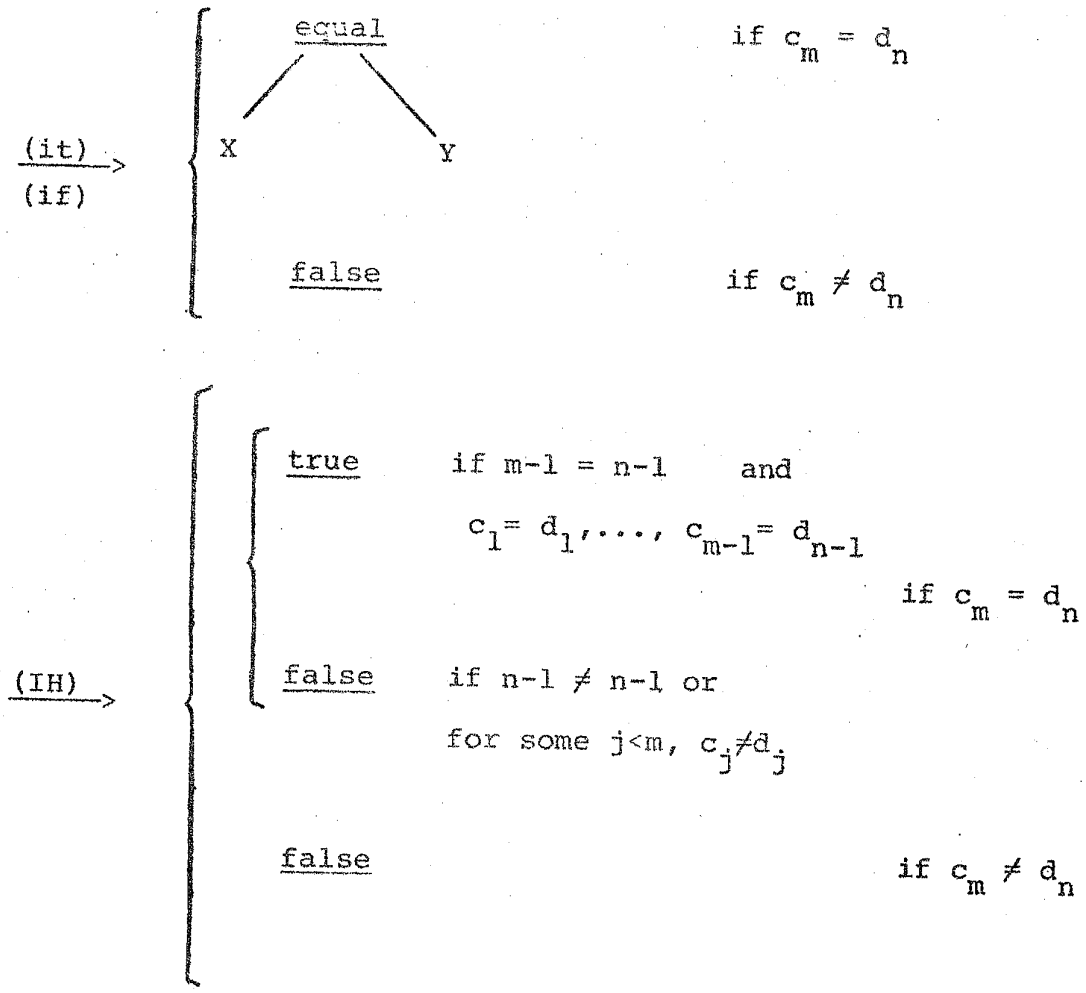
. subcase $n > 0$. Here both $m \neq 0$ and $n \neq 0$.

We have




(ecc)
if - then - else -
same
equal
false
c_m d_m
c_{m-1} cons
c_1 cons
null
cons d_{n-1}
cons d_1
null
X
Y


(S)
if - then - else -
true
equal
false
X Y
if $c_m = d_n$
if - then - else -
false
equal
false
X Y
if $c_m \neq d_n$


(it)
(if)
equal
X
Y
if $c_m = d_n$
false
if $c_m \neq d_n$
(IH)
true
if m-1 = n-1 and
$c_1 = d_1, \ldots, c_{m-1} = d_{n-1}$
if $c_m = d_n$
false
if n-1 ≠ n-1 or
for some j<m, $c_j \neq d_j$
false
if $c_m \neq d_n$