
# PUC

SERIES: MONOGRAFIAS EM CIÊNCIA DA COMPUTAÇÃO
Nº 2/83

# ON MULTI-STEP METHODS FOR STEPWISE CONSTRUCTION OF ALGEBRAIC SPECIFICATIONS FOR DATA BASE APPLICATIONS

by

Paulo A. S. Veloso
and
A. L. Furtado

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 2/83

Series Editor: A. L. Furtado January 1983

# ON MULTI-STEP METHODS FOR STEPWISE CONSTRUCTION OF ALGEBRAIC SPECIFICATIONS FOR DATA BASE APPLICATIONS*

by

P. A. S. Veloso

and

A. L. Furtado

* Research partly sponsored by FINEP and CNPq.

Abstract

A multi-step methodology for the formal specification of data base applications, entirely within the algebraic approach to abstract data types, is presented. Towards this end the concept of traces is extended to several levels, which gives a useful tool to obtain formal specifications in a systematic and constructive way. In addition, the concept of traces has familiar simple analogues. The presentation is based on a simplified example of a data base, which is successively specified in three formats: procedural notation, systems of term-rewriting rules and conditional equations.

Keywords:

abstract data types, data bases, formal specifications, traces, specification methodologies, trace levels, equational specifications, rewrite rules, procedural specifications.

RESUMO

Apresenta-se uma metodologia multi-passo, inteiramente contida no enfoque algébrico de tipos abstratos de dados, para a especificação formal de aplicações de bancos de dados. Para isso, o conceito de traços é estendido a vários níveis, o que fornece uma ferramenta útil para a obtenção sistemática e construtiva de especificações formais, além de estabelecer analogias com noções simples e familiares. A apresentação é baseada em um exemplo simples, que é sucessivamente especificado em três formatos: notação procedural, sistemas de regras de reescrita de termos e equações condicionais.

PALAVRAS CHAVES:

Tipos abstratos de dados, bancos de dados, especificações formais, metodologias para especificação, níveis de traço, especificações equacionais, regras de reescrita, especificações procedurais.

CONTENTS

## 1. Introduction

This paper proposes a multi-step methodology, entirely within the algebraic approach, for obtaining formal specifications of abstract data types and, in particular, data base applications. The main tool proposed to aid in the systematic construction of these algebraic specifications is the concept of traces extended to several levels.

The algebraic approach to abstract data types has been widely advocated as a useful tool for the formal specification of data structures and, in particular, data bases. However, formal specifications in general are not praised for their legibility or ease of construction. In fact, due to the difficulties in obtaining an algebraic specification directly from a model (given formally or, worse, informally) some methodologies have been suggested. Canonical term algebras, used to verify the correctness of specifications [GTW], have been used also as an aid in constructing an algebraic specification [PV], being more helpful if used in conjunction with rewrite rules [Ve]. Helpful as they are, these methodologies still leave room for improvement. On the other hand, some multi-step methods have been proposed [EF,VCF]. These methods usually involve intermediate formalisms other than the algebraic one, which is one of their main disadvantages. For instance, one such method [VCF] involves specifications by means of logical axioms, abstract models, pre- and post-conditions, in addition to the algebraic one.

We propose here a multi-step methodology where, in contrast, each step can be carried out entirely within a single formalism, in this case, the algebraic formalism itself. Their main advantages stem from being both multi-step and within a single formalism. The usage of a single formalism avoids the problem of having to translate from a formalism to another one. Rather, and that is where the multi-step aspect comes in, what we have to do is to refine specifications within a single formalism. This feature of stepwise refinement is, of course, a major advantage.

The algebraic approach views each object of a data type as (represented by) a variable-free term, so that the abstract data type is (the isomorphism class of) a homomorphic image of

the free algebra of terms, the specification pinpointing which homomorphic image. In particular, a data base state can be represented by a sequence of update operations capable of generating it. One may regard such a term as a trace or log of how the data base has actually evolved on its way to the present state. As such, these terms may contain some extra information, which may be redundant if one is interested in the state per se rather than in a particular way of generating it.

The proposed methodology starts at a level where all ground terms are taken as representatives for states and gradually proceeds via a series of intermediate levels until reaching the desired level, e.g., one with a unique representative for each state. On the intermediate levels the specifications progress towards smaller sets of representatives by considering fewer sequences of updates. As a consequence, the specifications gradually shift their main orientation from queries to updates. Nevertheless, each level specification is consistent and sufficiently complete. Furthermore, at each level, we can employ three kinds of algebraic formalisms to express the specifications: (conditional) equations, systems of term-rewriting rules, and procedural notation, the latter leading to executable specifications.

The structure of the paper is as follows. The next section reviews some mathematical preliminaries in order to fix the terminology and notation. Then section 3 introduces an illustrative example to be used throughout the paper. This simple example is employed in section 4 to convey the basic ideas of trace levels. Section 5 formalizes trace levels in terms of canonical term algebras and presents some of their properties. Section 6 illustrates these ideas by presenting semi-formal specifications for various trace levels of our running example, which are translated into the procedural formalism in section 7. In section 8 we consider and illustrate the methodology as couched in the context of systems of term-rewriting rules. These ideas are further developed in section 9, where we outline the process of deriving an equational specification, again using our running example. Finally, section 10 concludes with some general remarks. This structure is modular in that one can skip sections 2 and 5 on a first reading in order to get only the main intuitive ideas.

## 2. Preliminaries

We shall employ the usual notation and terminology for abstract data types [GTW,GH]. For a neat presentation we refer to [Pa], whose main concepts of interest here we outline in the sequel.

A signature L consists of:

. a non-empty set S of sorts

. a set $\Sigma$ of operation symbols

. a profile declaration $\Pi:\Sigma \to S^* \times S$ assigning to each operation symbol $\sigma \in \Sigma$ its functionality, noted $\sigma: s_1 \ldots s_n \to s$.

Terms are defined as usual and collected according to their target sorts. In general, we call such families of sets indexed by S an S-set.

An L-algebra A consists of an assignment of non-empty domains to the sorts of S and of operations to operation symbols respecting their profiles. In other words, $A = (A, \Sigma^A)$ where A is an S-set of domains $A_s \neq \emptyset$ for $s \in S$ and for $\sigma \in \Sigma$ with profile $\Pi(\sigma) = (s_1 \ldots s_n, s)$, $\sigma^A: A_{s_1} \times \ldots A_{s_n} \to A_s$.

The S-set of all (ground) terms T of L can be given a natural structure of L-algebra, called the term-algebra $T$ of L, which is initial in the category of L-algebras (with L-homomorphisms) [GTW]. In fact, given an algebra A the mapping assigning to each $t \in T$ its denotation (or value) $t^A$ in A is the unique homomorphism $h^A$ of $T$ into A. We call A finitely generated [WB] iff this denotation map $h^A$ is onto A. Then, by the isomorphism theorem [Gr], the assignment $A \mapsto \equiv [A]$ (where $t_1 \equiv t_2[A]$ iff $t_1^A = t_2^A$) gives a one-to-one correspondence between the finitely generated algebras and the complete lattice of congruences on $T$.

The algebra $T$ of terms has the advantage of having syntactical domains. Other such algebras will also be of interest here.

A canonical form F is an S-set of terms (called canonical terms) that is closed under the formation of subterms, i.e. whenever $\Pi(\sigma) = (s_1 \ldots s_n, s)$ and $\sigma t_1 \ldots t_n \in F_s$ then $t_1 \in F_{s_1}$, ..., $t_n \in F_{s_n}$. A canonical term algebra (cta, for short) is an

algebra $C$ whose domains constitute a canonical form and their operations consist of syntactical manipulations on canonical terms in the sense that whenever $\Pi(\sigma) = (s_1 \ldots s_n, s)$ and $\sigma\ t_1 \ldots t_n \in C_s$ then $\sigma^C[t_1,\ldots,t_n] = \sigma\ t_1 \ldots t_n$.

Canonical forms have their origin in the normal forms of term rewriting systems [HO]. A cta adds to the advantage of having syntactical domains that of ease in carrying out inductive arguments, since being "closed under the formation of subterms" generally is what one needs in the inductive step. Notice that a cta is not a subalgebra of the algebra of terms $T$, but we have the following useful property:

Lemma [GTW] - If $C$ is a cta and $h^C$ is the denotation homomorphism from $T$ to $C$, then $h^C$ is onto and for each $t \in C$ $h^C(t) = t$.

As said before, a finitely generated algebra is characterized by a congruence on $T$. In order to describe such congruences we resort to some specification formalism, which consists of finite number of schemas involving terms with variables and generating the desired congruence.

In many cases it is convenient to single out certain sorts as of special interest (cf. the sort of interest TOI of [GHI), viewing the others as, say, parameters. In particular for data bases, a domain of special interest is that consisting of data base states or instances. Operations with values in this domain are updates, the others - with values in other sorts - are queries.

This leads to the notion of hierarchical signature, which is a signature as before with a specified sub-signature called basic sub-signature, consisting of basic sorts and basic operations whose profiles involve only basic sorts. The terms generated by the basic operations are called primitive terms. Notice that the set $P_s$, of primitive terms of a basic sort $s$ is a subset of $T_s$. Call an operation symbol whose target sort is basic an external operation.

A hierarchical data type consists of a hierarchical signature together with an algebra, $B$, called basic algebra, for the basic sub-signature. This basic algebra induces a congruence $=_P$, called primitive congruence, on $P$.

A hierarchical algebra is an algebra of the signature, whose reduct to the sub-signature coincides with the basic algebra. The congruences corresponding to hierarchical algebras with a given basic algebra B form a complete sublattice of the lattice of all congruences on $T$.

Two important properties of a congruence $\theta$ on $T$ are the following:

- $\theta$ is consistent iff any class of $\theta$ contains at most one class of $=_P$, i.e. whenever $(p,p') \in \theta$, for $p,p' \in P$, also $p =_P p'$.
- $\theta$ is sufficiently complete iff any class of $\theta$ contains at least one class of $=_P$, i.e. for any term $t$ of a basic sort there exists a primitive term $p \in P$, such that $(t,p) \in \theta$.

The above terminology is generally employed in connection with specifications.

We shall be dealing with three specification formalisms: equational [GTW], rewriting systems [HO] and procedural notation [FV]. A set E of equations (pairs of terms with variables) describes the least congruence $\equiv[E]$ on $T$ containing all ground instances of the equations [GTW]. Simflarly, a rewriting system R consisting of rewrite rules (ordered pairs of terms with variables) describes the relation $\equiv[R]$ on T (relating two terms that can be converted into a common one), which is a congruence on $T$ if R is confluent (i.e. has the Church-Rosser property [HO]). A procedural specification is a CLU-like cluster [Li], consisting of a symbol-manipulating procedure for each operation symbol. It can be regarded as a deterministic implementation of a rewriting system obtained by super-imposing some order of application for the rewrite rules [FV].

A specification $\Gamma$ in one of the above formalisms generates a syntactical congruence $\equiv[\Gamma]$ on $T$ as follows: $t \equiv t'[\Gamma]$ iff the equality $t = t'$ can be derived from $\Gamma$.

Given an algebra A, by means of its congruence $\equiv[A]$, a specification $\Gamma$ is (see Fig. 1)

- correct for A iff $\equiv[\Gamma] \subseteq \equiv[A]$, i.e. any equality $t = t'$ derivable from $\Gamma$ is true in A.

. complete for A iff $\equiv[A] \subseteq \equiv[\Gamma]$, i.e. any equality $t = t'$ holding in A is derivable from $\Gamma$.

The specification problem (in a given formalism, as above) for a given algebra A consists of finding a specification $\Gamma$ (in the given formalism) that is both correct and complete with respect to A.

In the following sections we shall show how the notion of traces, in particular several levels of traces, can be used in a methodology for the stepwise construction of specifications for abstract data types.

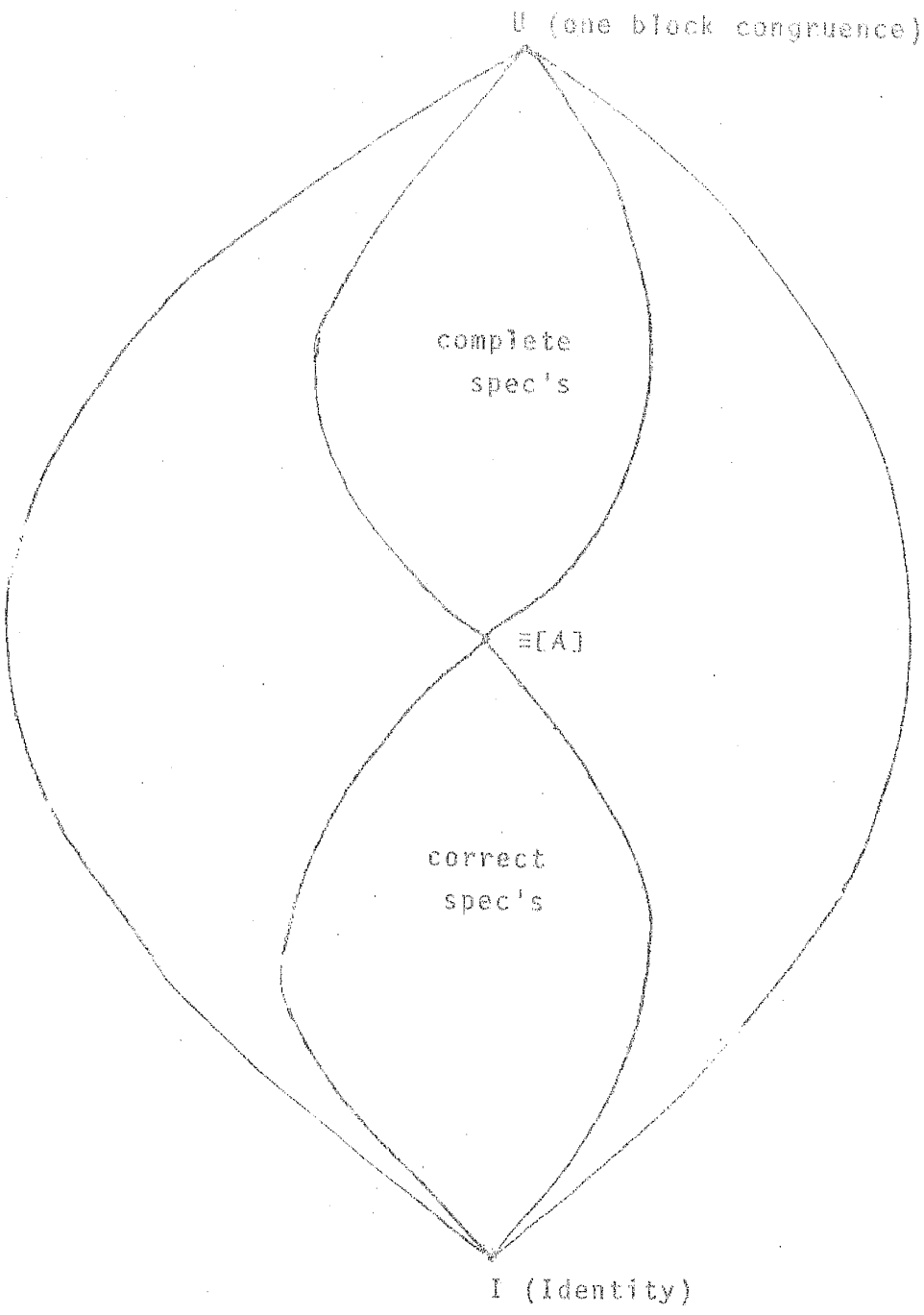


Fig. 1: Correct and complete specifications with respect to A.

## 3. An Illustrative Example

As a simple example to illustrate the discussion we shall use the data base of a company, call it Acme, marketing a machine. Acme can either lease or sell a machine to a customer. In both cases the customer will use the machine, but only in the latter case will he own it. If a machine has been leased to a customer he may later decide to buy it or else he may choose to return it to Acme. For simplicity sake we shall assume that there is only one kind of machine and at any time a customer will have at most one machine.

The operation symbols here are the words underlined in the preceding paragraph together with the usual update phi which initializes the data base to an "empty" state. More specifically, we have 3 sorts state, customer and Bool(ean), the TOI being state and the other two being basic sorts.

Operation symbol phi has profile ($\lambda$, state); it is a constant for initialization. On the other hand lease, sell and return all have profile (customer state, state), thus being updates. Also, uses and owns have profile (customer state, Bool); they are queries. In addition the basic sorts have some basic operation symbols. Bool has two constant symbols True and False and customer is assumed to be supplied with operation symbols allowing the generation of variable-free terms as names of customers. Finally, we also have operation symbols = and $\neq$ of profile (customer customer, Bool) to compare these customers.

The above explicitation is a description of our hierarchical signature L. So far, we have a formal description of the syntax of our example. The meaning of the operations has been given only informally in natural language.

It is assumed that each state can be identified by the results it yields to the queries. Also in the initial state all queries yield False and the only way to reach a state where a query yields True is by applying some sequence of updates.

So, the effect of an update is to cause changes in the answer to queries. Conversely certain answers to queries may be preconditions for an update; if the preconditions fail the ap-

plication of the update will not change the state.

A specification for the updates of our example in terms of preconditions and effects is as follows:

t := phi
  preconditions : none
  effects : $\forall c \neg uses(c,t) \wedge \neg owns(c,t)$

t := lease(c,s)
  preconditions : $\neg uses(c,s)$
  effects : $uses(c,t)$

t := sell(c,s)
  preconditions : $\neg owns(c,s)$
  effects : $uses(c,t) \wedge owns(c,t)$

t := return(c,s)
  preconditions : $uses(c,s) \wedge \neg owns(c,s)$
  effects : $\neg uses(c,t)$

This specification, together with the explicitation of the underlying assumptions [VCF] (such as the well-known frame assumption: properties not explicitly mentioned are preserved under the application of an update), constitute a formal description of the semantics of our example. It is not an algebraic specification, though.

## 4. The Idea of Trace Levels

In the algebraic approach each data base state is denoted by a ground term representing a sequence of update operations capable of generating it. One may regard such a term as a log or trace of how the data base has been manipulated to attain this particular state. As such, these terms may contain extra information that can be discarded if one is interested in the state per se rather than in a particular way of generating it.

In order to clarify these ideas, let us assume that our example data base is in the initial empty state and the following sequence of transactions is performed:

1. a machine is leased to customer D;
2. a machine is leased to customer B;
3. D returns a machine;
4. customer C buys a machine;
5. customer B buys a machine;
6. C returns a machine;
7. a machine is leased to customer A;
8. a machine is leased to customer B.

The list of (the symbols of) the above 9 updates (beginning with phi and ending with the last lease) would be a log of this sequence of transactions. It is often more convenient, however, to represent this trace in an "applicative" format as

lease(B,lease(A,return(C,sell(B,
sell(C,return(D,lease(B,lease(D,phi))))))))). (1)

The execution of these updates will cause the data base to evolve from the initial empty state, through a series of intermediate states, to a state P where customers A, B and C use machines and B and C own machines. This state can be represented by the two sets $U_p = \{A,B,C\}$ and $O_p = \{B,C\}$. Similarly, each intermediate state s can be described by its set $U_s$ of customers using machines and its set $O_s$ of customers owning machines (notice that $O_s \subseteq U_s$ is a static integrity constraint of our example). The next table shows this evolution.

| S | $U_S$ | $O_S$ |
|---|---|---|
| 0 | ∅ | ∅ |
| 1 | {D} | ∅ |
| 2 | {B,D} | ∅ |
| 3 | {B} | ∅ |
| 4 | {B,C} | {C} |
| 5 | {B,C} | {B,C} |
| 6 | {B,C} | {B,C} |
| 7 | {A,B,C} | {B,C} |
| 8 | {A,B,C} | {B,C} |

Some points are worth remarking in connection with this trace and corresponding evolution.

(a) States 5 and 6, having identical sets O and U, are the same. Likewise for states 7 and 8. If we delete from trace (1) the updates return(C,.) and lease(B,.) causing these repetitions we obtain the evolution:

| update | U | O |
|---|---|---|
| phi | ∅ | ∅ |
| lease(D,.) | {D} | ∅ |
| lease(B,.) | {B,D} | ∅ |
| return(D,.) | {B} | ∅ |
| sell(C,.) | {B,C} | {C} |
| sell(B,.) | {B,C} | {B,C} |
| lease(A,.) | {A,B,C} | {B,C} |

corresponding to the repetition-free trace

lease(A,sell(B,sell(C,return(D,lease(B,lease(D,phi)))))) (2)

denoting the same final state P as (1).

(b) In going from state 2 to 3, the set O decreases from {B,D} to {B}. This is due to the fact that the update return(D,.) cancelled the effects of a previous lease(D,.). If we eliminate these updates from trace (1) we obtain the following evolution:

| update | U | O |
|---|---|---|
| phi | ∅ | ∅ |
| lease(B,.) | {B} | ∅ |
| sell(C,.) | {B,C} | {C} |
| sell(B,.) | {B,C} | {B,C} |
| return(C,.) | {B,C} | {B,C} |
| lease(A,.) | {A,B,C} | {B,C} |
| lease(B,.) | {A,B,C} | {B,C} |

corresponding to the following non-decreasing (albeit not repetition-free) trace denoting the same state P:

lease(B,lease(A,return(C,sell(B,sell(C,lease(B,phi)))))) (3)

(c) In state 2 we have $B \in O$ because of the update lease(B,.) and subsequently we have in state 5 both $B \in U$ and $B \in O$ because of update sell(B,.). So, the effects of sell(B,.) subsume those of the previous lease(B,.). Now, replace in trace (1) the first lease(B,.) by sell(B,.) to obtain the trace:

lease(B,lease(A,return(C,sell(B,
sell(C,return(D,sell(B,lease(D,phi))))))))) (4)

which is subsumption-free (but neither repetition-free nor non-decreasing). The evolution corresponding to (4) is the same as (1) except that states 2 and 3 now have O = {B} and state 4 has O = {B,C}.

(d) We can view the aim of executing the original sequence of updates as attaining a state P where $U_p$= {A,B,C} and $O_p$= {B,C}. From this viewpoint traces (1) through (4) achieved this aim in a somewhat roundabout way.If we want to increase (U,O) from (∅,∅) to ({A,B,C},{B,C}) via few intermediate states we may consider the following evolution:

| update | U | O |
|---|---|---|
| phi | ∅ | ∅ |
| sell(B,.) | {B} | {B} |
| sell(C,.) | {B,C} | {B,C} |
| lease(A,.) | {A,B,C} | {B,C} |

corresponding to the trace:

lease(A,sell(C,sell(B,phi))) (5)

If we eliminate from trace (5) any one of the updates we fail to achieve the same final state P. So, we call (5) a reduced trace.

(e) The effect of lease(B,.) at a state is the insertion of B into the corresponding set U. Likewise lease(C,.) causes the insertion of C into U. So, lease(B,.) and lease(C,.) commute, the net effect of their execution in any order being the insertion of both B and C into U. This gives us the freedom of reordering occurrences of the same operation symbol according to their customer parameters. We can choose an order among customer names (say A < B < C < D) and reorder adjacent occurrences of the same operation symbol so that A is more external than B, and so forth. For instance, trace (2) is already ordered according to this criterion. Ordered traces corresponding to the other ones are:

- corresponding to (1):
  lease(A,lease(B,return(C,sell(B,sell(C,
  return(D,lease(B,lease(D,phi))))))))) (6)
- corresponding to (3):
  lease(A,lease(B,return(C,sell(B,sell(C,
  lease(B,phi)))))) (7)
- corresponding to (4):
  lease(A,lease(B,return(C,sell(B,sell(C,
  return(D,sell(B,lease(D,phi))))))))) (8)
- corresponding to (5):
  lease(A,sell(B,sell(C,phi))) (9)

(f) Still other possible trace levels are combinations of the preceding ones. For instance, an increasing trace (i.e. repetition-free and non-decreasing) is:

lease(A,sell(B,sell(C,lease(B,phi)))) (10)

causing the evolution:

| O | U |
|---|---|
| ∅ | ∅ |
| {B} | ∅ |
| {B,C} | {C} |
| {B,C} | {B,C} |
| {A,B,C} | {B,C} |

where each transition causes a strict increment in O or U.

(9) Of special interest are trace levels where each state has a unique representative, for then we have a one-to-one correspondence between states and traces denoting them. A systematic way to achieve this uniqueness consists of proceeding via a series of traces, each one "closer" to uniqueness than the preceding one. For instance:

- starting at the actual trace level, containing traces such as (1),
- pass to the repetition-free level, with traces as (2),
- then move on to the increasing level, with traces as (10),
- and proceed to "more definite" levels by adding successively the conditions of being subsumption-free, reduced, etc. and finally ordered.

## 5. Trace Levels and Algebra

A subterm of a trace corresponds to a past portion of a log. For instance, trace (5) is a reduced trace and its subterm sell(B,phi) is a reduced trace, as well. Moreover sell(B,phi) is a reduced trace for an intermediate state in the evolution corresponding to trace (5).

So it is natural to define a trace level as a canonical form and a trace algebra as a canonical term algebra (cta). Thus, a trace level is a set of representatives for the congruence classes of the corresponding trace algebra.

Trace levels can be naturally ordered by inclusion. Also, it is easy to see that the property of being a canonical form is closed under arbitrary unions and intersections. Thus we have (see Fig. 2)

Proposition The set of all trace levels of a signature L, under inclusion, forms a complete lattice. Its lub (supremum) is the actual trace level and its glb (infimum) is the empty canonical form.

Notice that we have admitted an empty canonical form but it corresponds to no cta. The least trace levels corresponding to cta's are those consisting of a single term for each sort. Moreover, a given congruence can have in general many sets of representatives. In fact the proof of

Lemma [GTW] Any finitely generated algebra is isomorphic to a cta.

indicates some ways of choosing canonical representatives. In addition this lemma shows that we can always work with trace algebras without losing any finitely generated algebra.

In using trace levels as a tool for methodical specification we are interested only in canonical forms that can be refined so as to represent a given algebra A. In other words, we are interested in cta's C with $\equiv[A] \supseteq \equiv[C]$. Now the corresponding canonical forms are no longer closed under intersection. But we still have (see Fig. 3)

Theorem The set of trace levels of cta's with congruences included in a given congruence $\theta$ on $T$ is a complete upper sub-semi-

lattice of the lattice of all trace levels. Its lub is the actual trace level and its minimal elements are the trace levels of the cta's isomorphic to $T/\theta$.

In section 4 we have introduced the idea of trace levels by means of an illustrative example. Some of the trace levels presented therein have wide applicability, therefore deserving general definition and comments.

First the actual trace level is the canonical form consisting of all (syntactically correct) (ground) terms without any further restriction. It corresponds to the term algebra $T$.

The repetition-free level deserves further clarification, in view of a distinction not clearly stressed in section 4. A trace level free from adjacent repetitions is an S-set P of terms such that whenever $\sigma\ t_1 \ldots t_i \ldots t_n \in P_s$ and $t_i \in P_s$ then their values in the given algebra A are distinct: $t_i^A \neq \sigma^A[t_1^A, \ldots, t_n^A]$. A trace level free from (arbitrary) repetitions is an S-set Q satisfying the stronger requirement that whenever a term $t \in Q_s$ has one of its proper subterms $\tilde{t}$ also in $Q_s$ then $t^A \neq \tilde{t}^A$.

Now a trace level R is said to be reduced iff whenever a term t is in $R_s$ and one of its subterms t' has the form $\sigma\ t_1 \ldots t_i \ldots t_n$ with $t_i \in R_s$ then $t^A \neq t_i^A$.

Finally, a trace level U is said to be on the unique-representative level iff for each $a \in A$ there exists exactly one trace $t \in U$ denoting a, i.e. with $t^A = a$.

The following implications are easily seen to hold:

unique-representative $\Rightarrow$ reduced $\Rightarrow$ free from adjacent repetition.

Moreover, they cannot be reversed, in general. Also, notice that while the property of being a canonical form is purely syntactical, this is no longer true for trace levels. A trace level is free from adjacent repetitions, reduced, etc. only with respect to a given algebra A.

The preceding discussion of trace levels and trace algebras concerns finitely generated algebras in general. In the case of hierarchical data types we are given a basic algebra B. We can then relativize our concepts to the non-basic sorts, as follows. For the basic sorts we choose one primitive term to represent each equivalence class of the primitive congruence. For the non-basic

sorts we proceed as before. Thus we obtain hierarchical trace levels and algebras. More formally, a hierarchical trace algebra is a cta, whose reduct to the basic sub-signature is isomorphic to the basic algebra. Also, we call a hierarchical trace level actual, non-decreasing, reduced, etc. iff viewed as a trace level it satisfies the corresponding restriction.

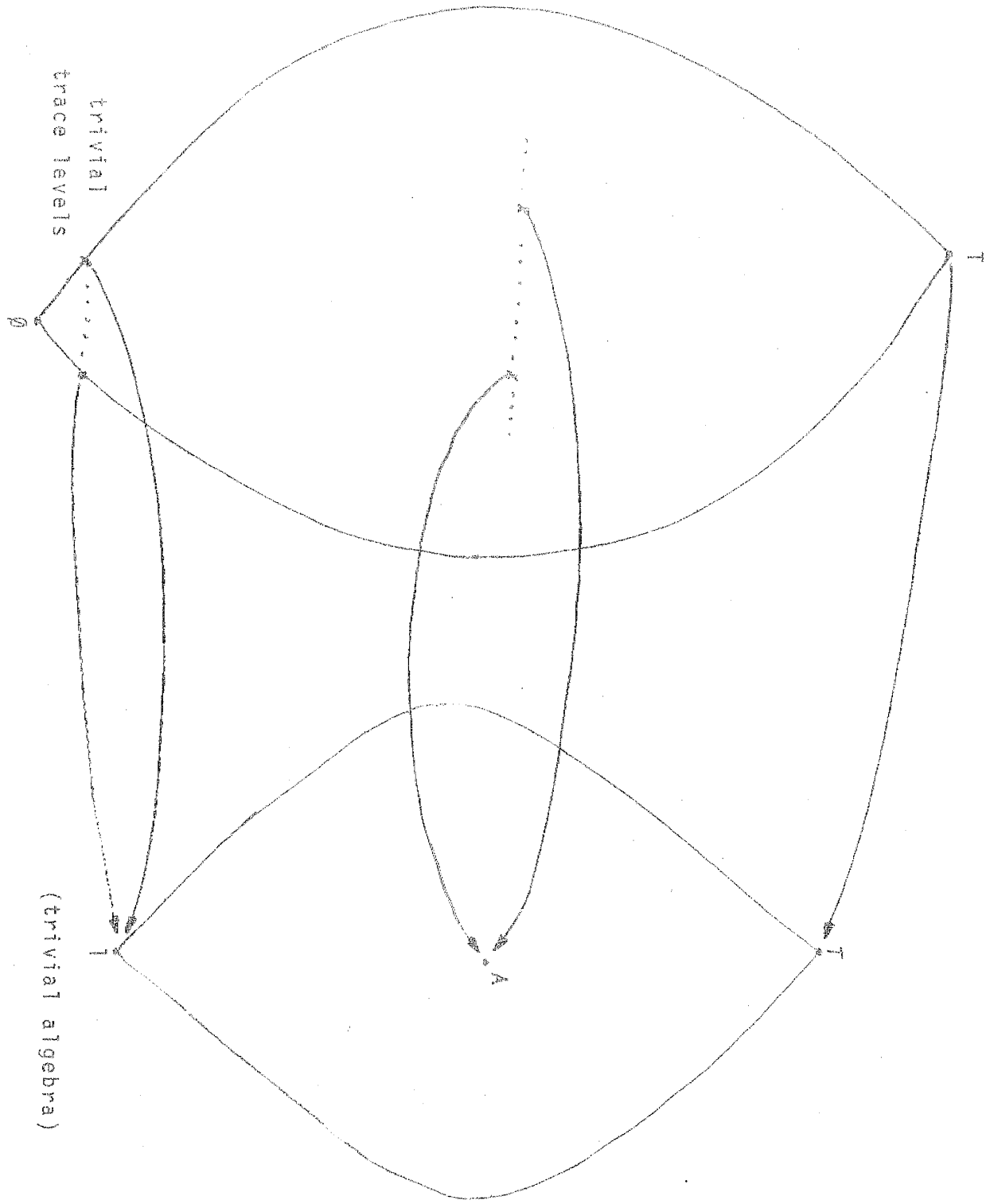


Fig. 2: Correspondence between the lattices of trace levels and of finitely generated algebras.

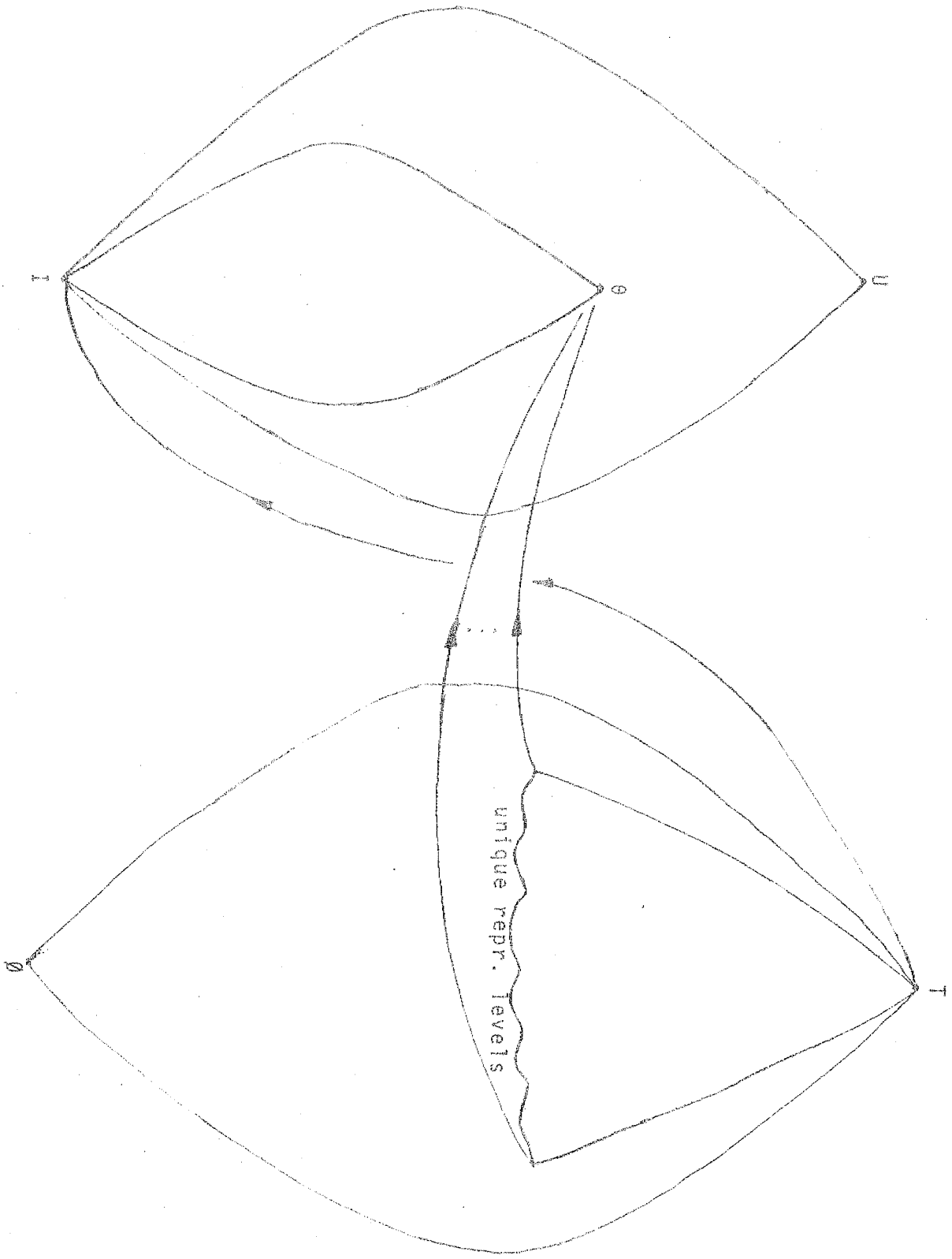


Fig. 3: Sublattice of congruences included in θ and corresponding upper sub-semilattice of trace levels.

## 6. Trace Level Specifications

We shall now use our running example to illustrate various hierarchical trace level specifications and their usage. We shall describe, in a semi-formal way, the following cta's:

- the cta $T$ of the actual (hierarchical) trace levels;
- the cta $P$ of the repetition-free (hierarchical) level;
- the cta $D$ of the reduced (hierarchical) level;
- the cta $U$ of the unique-representative (hierarchical) level.

It will become clear that what we call, for short, repetition-free level should more properly be called "free from adjacent repetitions", as defined in section 5.

### 6.1. Actual (hierarchical) trace level

On this level, the traces of sort customer are all (syntactically correct) terms, representing the actual sequence of all the updates actually invoked in their chronological order. This reminds one of the so-called audit trails. Indeed, for auditing or for statistical purposes these actual traces may be useful. As an example consider:

lease(C,return(A,return(B,lease(C,
sell(B,lease(A,lease(B,phi)))))))  (11)

A semi-formal specification for this level consists of:

.....

$\text{lease}^T[c,t] = \text{lease}(c,t)$

$\text{sell}^T[c,t] = \text{sell}(c,t)$

.....

$$\text{owns}^T[c,t] = \begin{cases} \text{True} & \text{if t contains sell}(C,\ldots) \\ \text{False} & \text{otherwise} \end{cases}$$

Of course, this specifies the cta $T$ of all terms of sort state. (Bool(ean) is assumed to have the constants True and False as traces and the sort customer is assumed to have constants as representatives for customers.) Notice that the specification is sufficiently complete (one can determine the results of queries; for instance, with t denoting trace (11), as it contains sell(B,...), we have $\text{owns}^T[B,t] = \text{True}$), and correct with respect to the intended

model but is not complete. Indeed, the value of trace (11), as any other trace, is itself on this level, in the sense

$\underline{lease}^{T}[C,\underline{return}^{T}[A,\ldots,\underline{lease}^{T}[B,\underline{phi}]\ldots]] =$

$= \underline{lease}(C,\underline{return}(A,\ldots,\underline{lease}(B,\underline{phi})\ldots))$

## 6.2. Repetition-free (hierarchical) trace level

Here, a trace consists only of the operation symbols that actually caused state changes. For instance, a repetition-free trace corresponding to (11) is

$$\underline{return}(A,\underline{lease}(C,\underline{sell}(B,\underline{lease}(A,\underline{lease}(B,\underline{phi}))))) \qquad (12)$$

Again, this level reminds one of logs kept, in this case, for recovery purposes.

A semi-formal specification for this level is as follows:

.....

$$\underline{lease}^{P}[c,t] = \begin{cases} \underline{lease}(c,t) & \text{if } \underline{uses}^{P}[c,t] = \underline{False} \\ t & \text{otherwise} \end{cases}$$

$$\underline{return}^{P}[c,t] = \begin{cases} \underline{return}(c,t) & \text{if } \underline{uses}^{P}[c,t] = \underline{True} \\ & \text{and } \underline{owns}^{P}[c,t] = \underline{False} \\ t & \text{otherwise} \end{cases}$$

.....

$$\underline{uses}^{P}[c,t] = \begin{cases} \underline{True} & \text{if } t \text{ contains } \underline{sell}(c,\ldots) \\ & \text{or } t = u_1(c_1,u_2(c_2,\ldots,u_n(c_n, \\ & \underline{lease}(c,t'))\ldots)) \\ & \text{and whenever } u_i = \underline{return}, \text{ then} \\ & c_i \neq c \\ \underline{False} & \text{otherwise} \end{cases}$$

$$\underline{owns}^{P}[c,t] = \begin{cases} \underline{True} & \text{if } t \text{ contains } \underline{sell}(c,\ldots) \\ \underline{False} & \text{otherwise} \end{cases}$$

Again, this is a sufficiently complete specification, correct with respect to the intended model and it is a refinement of the preceding one.

On this level, the value of the trace (12) is itself, in the sense

$$\underline{return}^{P}[A,\underline{lease}^{P}[C,\ldots,\underline{lease}^{P}[B,\underline{phi}^{P}]\ldots]] =$$
$$= \underline{return}(A,\underline{lease}(C,\ldots,\underline{lease}(B,\underline{phi})\ldots))$$

as can be seen from the above specification. However the value of trace (11) can be obtained as follows. First, the specification gives

$$\underline{lease}^{P}[C,\underline{sell}^{P}[B,\underline{lease}^{P}[A,\underline{lease}^{P}[B,\underline{phi}^{P}]]]] =$$
$$\underline{lease}(C,\underline{sell}(B,\underline{lease}(A,\underline{lease}(B,\underline{phi}))))$$

Call this term t'. First, as t' contains $\underline{sell}(B,\ldots)$, we have $\underline{owns}^{P}[B,t'] = \underline{True}$, whence $\underline{return}^{P}[B,t'] = t'$. Now, as t' contains $\underline{lease}(A,\ldots)$ with no later $\underline{return}(A,\ldots)$, we have $\underline{uses}^{P}[A,t'] = \underline{True}$. Also $\underline{owns}^{P}[A,t'] = \underline{False}$. So $\underline{return}^{P}[A,t'] = \underline{return}(A,t')$. Now $\underline{uses}^{P}[C,\underline{return}(A,t')] = \underline{True}$, which implies $\underline{lease}^{P}[C,\underline{return}(A,t')] = \underline{return}(A,t')$. Hence the value of trace (11) on this level is trace (12). As this equality could not be derived on the actual trace level, we now have a proper refinement, i.e. $\equiv[P] \subsetneq \equiv[T]$.

## 6.3. Reduced (hierarchical) trace level

On the preceding levels the length of the traces increased with time. Now we keep only those operation symbols whose effects were not later cancelled or subsumed by others. (In this example this is enough to guarantee that the trace is reduced.) A reduced trace corresponding to (12) is

$$\underline{lease}(C,\underline{sell}(B,\underline{phi})) \qquad (13)$$

A semi-formal specification for this level is

.....

$$\underline{sell}^{D}[c,t] = \begin{cases} \underline{sell}(c,t'), \text{ where } t' \text{ is the result of removing any occurrence of } \underline{lease}(C,\ldots) \text{ from } t & \text{if } \underline{owns}^{D}[c,t] = \underline{False} \\ t & \text{otherwise} \end{cases}$$

$$\underline{return}^{D}[c,t] = \begin{cases} t & \text{if } \underline{owns}^{D}[c,t] = \underline{True} \\ & \text{or } \underline{uses}^{D}[c,t] = \underline{False} \\ \text{the result of removing } \underline{lease}(c,\ldots) \text{ from } t & \text{otherwise} \end{cases}$$

.....

$$\underline{uses}^{D}[c,t] = \begin{cases} \underline{True} & \text{if } t \text{ contains } \underline{lease}(c,\ldots) \text{ or } \underline{sell}(c,\ldots) \\ \underline{False} & \text{otherwise} \end{cases}$$

$$\underline{owns}^{D}[c,t] = \begin{cases} \underline{True} & \text{if } t \text{ contains } \underline{sell}(c,\ldots) \\ \underline{False} & \text{otherwise} \end{cases}$$

This specification is sufficiently complete, a proper refinement of the preceding one and still consistent with the intended model.

## 6.4. Unique-representative (hierarchical) trace level

Even on the reduced level a state can be represented by more than one trace. If, however, we decide to order the customers and the corresponding updates as illustrated in section 4, we obtain the desired uniqueness. But since we have on the reduced trace level only 2 updates with customers as parameters, we prefer to reorder the trace so that A appears before B, and so forth, independently of the associated updates. So, a trace corresponding to (13) on this level is:

$\underline{sell}(B,\underline{lease}(C,\underline{phi}))$

A semi-formal specification for this level is:

.....

$$\underline{sell}^{U}[c,t] = \begin{cases} \text{the result of removing any occurrence of } \underline{lease}(c,\ldots) \text{ and inserting } \underline{sell}(c,\ldots) \text{ in the appropriate position} & \text{if } \underline{owns}^{U}[c,t]=\underline{False} \\ t & \text{otherwise} \end{cases}$$

.....

$$\underline{uses}^{U}[c,t] = \begin{cases} \underline{True} & \text{if } t \text{ contains } \underline{sell}(c,\ldots) \text{ or } \underline{lease}(c,\ldots) \\ \underline{False} & \text{otherwise} \end{cases}$$

Now, as each state is represented by a unique trace, we have a correct and complete specification for the intended model, i.e. $U \cong A$.

This sequence of four hierarchical trace levels is illustrated in Fig. 4, where the value of traces (11), (12), (13) and (14) is shown on each level.

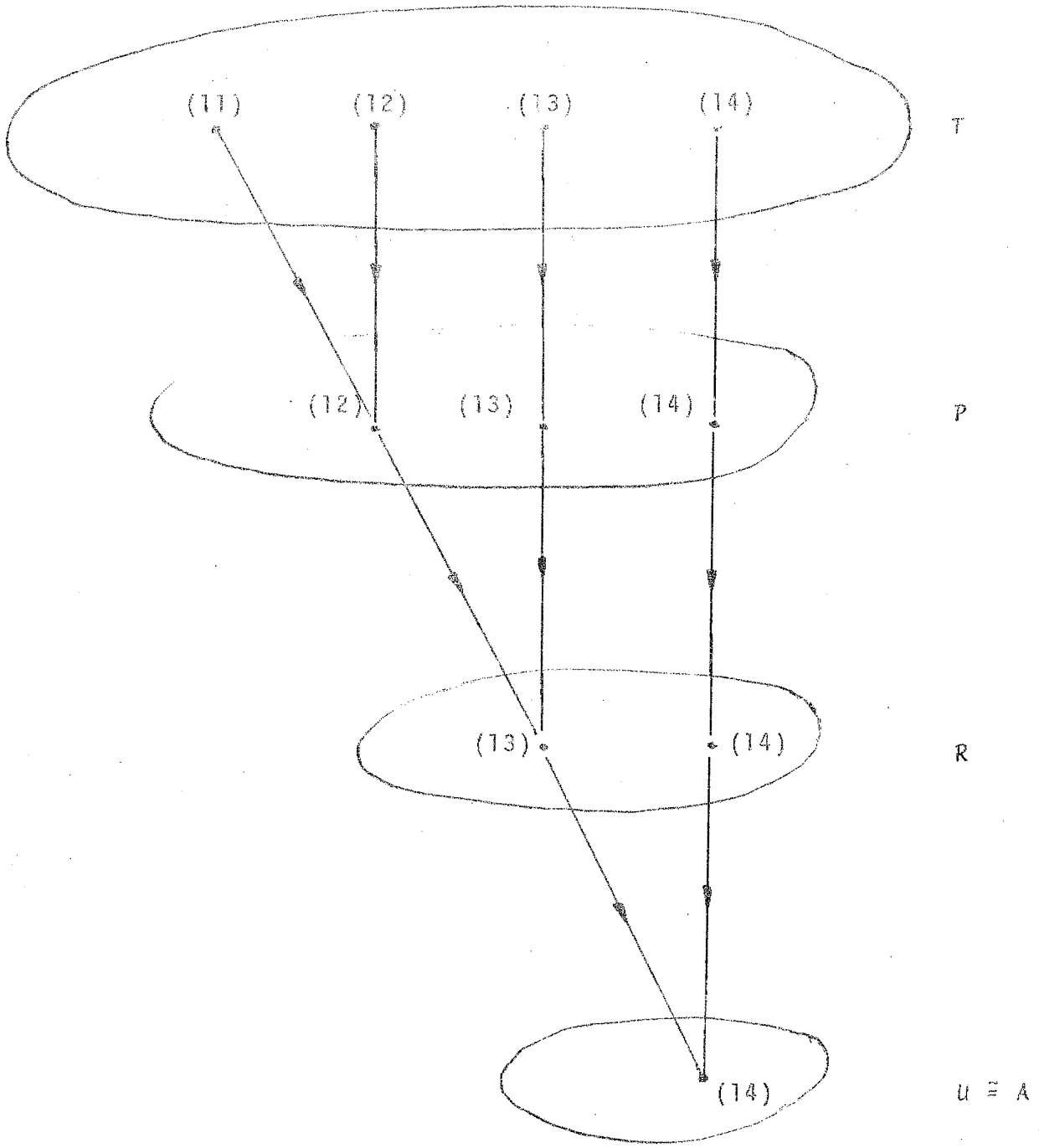


Fig. 4: Sort state of four hierarchical trace algebras

## 7. Procedural Specifications

As mentioned before, the execution of the operations of a cta involves inspecting and manipulating the traces. A procedural specification regards a trace as a sequence of symbols and "implements" the operations by means of procedures for symbolic manipulations.

Here we shall describe such procedures by means of a procedural notation [FV], whose main features are as follows. Each procedure (op) has a heading and a body. Within the latter, statements are sequentially executed and the value returned is that of the first expression on the right of a '⇒' whose left-hand side has the value True. The match statement is a case-like construct for pattern-matching, its value being that of the right-hand side of the expression whose left-hand side matches the trace.

### 7.1. Actual (hierarchical) trace level

For the actual trace level, the symbolic execution of an update is trivial; it suffices to add to the trace the symbol of the new operation. On the other hand, in order to know whether B uses a machine at the state denoted by (11) it is not enough to find return(B,...) (and answer False) neither can we answer True simply because we found lease(B,...). It is necessary to check whether the corresponding updates caused state changes and were not later cancelled.

A procedural specification for this level consists of procedures as the following:

```
.....................
op sell(c:customer, t:state) : state
   ⇒ sell(c,t)
endop
.....................
```

```
op uses(c:customer, t:state) : Bool
   var x:customer, s:state
   match t
     phi    ⇒ False
     lease(x,s) ⇒ if c = x then True
                  else uses(c,s)
     sell(x,s)  ⇒ if c = x then True
                  else uses(c,s)
     return(x,s)⇒ if c = x then owns(c,s)
                  else uses(c,s)
   endmatch
endop
```

## 7.2. Repetition-free (hierarchical) trace level

Now the execution of updates is still relatively simple, for any addition will be at the left end of the trace; we only have to take care in adding only those operation symbols whose preconditions for state change are satisfied. The execution of queries becomes slightly simpler than on the actual level. For instance, if the trace contains return(A,...) without later occurrences of lease(A,...) or sell(A,...), we can guarantee that at this state customer A does not use a machine.

A procedural specification for the repetition-free level consists of procedures with the following aspect:

```
op sell(c:customer , t:state) : state
   owns(c,t)  ⇒   t;
   ⇒ sell(c,t)
endop
```

```
op uses(c:customer , t:state  : Bool
   var x:customer , s:state
   match t
     phi  ⇒  False
     lease(x,s)  ⇒  if c = x then True
                    else uses(c,s)
     sell(x,s)   ⇒  if c = x then True
                    else uses(c,s)
     return(x,s) ⇒  if c = x then False
                    else uses(c,s)
   endmatch
endop
.............................
```

Notice that the body of the procedure sell has a precondition.

## 7.3. Reduced (hierarchical) trace level

Now the execution of updates becomes somewhat more complex, as some manipulation within the trace may be necessary. On the other hand, the execution of queries becomes simpler as traces are more "compact" and contain no "negative" updates.

Procedures for the reduced trace level have the following aspect:

```
.............................
op sell(c:customer , t:state) : state
  var x:customer , s:state
  owns(c,t)  ⇒  t;
  ¬ uses(c,t)  ⇒  sell(c,t);
  match t
    lease(x,s)  ⇒  if c = x then sell(c,s)
                   else lease(x,sell(c,s))
    sell(x,s)   ⇒  sell(x,sell(c,s))
  endmatch
endop
.............................
op uses(c:customer , t:state) : Bool
  var x:customer , s:state
  match t
```

```
    phi    ⇒   False
    lease(x,s)  ⇒  if c = x then True
                   else uses(c,s)
    sell(x,s)   ⇒  if c = x then True
                   else uses(c,s)
  endmatch
endop
```

## 7.4. Unique-representative (hierarchical) trace level

On this level the execution of updates involves more internal manipulations than on the preceding one, since the addition of an operation symbol is to be performed at an appropriate position. The execution of queries, however is just as simple; in fact they can become more efficient if one takes advantage of the ordering.

A specification for this trace level in our procedural notation and employing the order '<' among customer names looks like:

```
.....................................
op sell(c:customer , t:state) : state
  var x:customer , s:state
  owns(c,t)  ⇒   t ;
  match t
    phi   ⇒   sell(c,t)
    lease(x,s)  ⇒   if x = c then sell(x,s)
                    else if c < x then sell(x,t)
                         else lease(x,sell(c,s))
    sell(x,s)   ⇒   if c < x then sell(c,t)
                    else sell(x,sell(c,s))
  endmatch
endop
.....................................
op uses(c:customer , t:state) : Bool
  var x:customer , s:state
  match t
    phi   ⇒   False
    lease(x,s)  ⇒   if c = x then True
                    else if c < x  then False
                         else uses(c,s)
```

```
sell(x,s)  ⇒  if c = x then True
              else if c < x then False
                   else uses(c,s)
  endmatch
endop
```

A procedural specification actually converts a sequence of operations into the corresponding trace. For instance, consider

```
lease(C,return(A,return(B,lease(C,sell(B,
lease(A,lease(B,phi))))))))                    (15)
```

If we execute expression (15) with the procedures of the repetition-free level we obtain as result trace (12),whereas the same expression (15) executed on the reduced trace level yields trace (13). On the other hand the result of executing

```
return(A,lease(C,sell(B,lease(A,lease(B,phi)))))
```

on the unique-representative level is

```
sell(B,lease(C,phi))
```

Some remarks concerning these specifications are in order. First, notice that, as we progress from the actual trace level to that of unique representatives, the syntactical complexity of the procedural specifications shifts from queries towards updates. Also the cluster-like module of a given level will generate exactly the traces of this level.

It is worthwhile mentioning that a procedural specification is not only formal but also executable. This allows the designer to experiment with the specification to see whether the original intentions were actually captured before committing to the costly and arduous task of machine implementation. Besides, it is relatively straightforward to translate a procedural specification into actual programs written in some symbol-manipulation language such as SNOBOL [FV].

## 8. Rewrite Rules

As mentioned, a procedural specification can be regarded as a device for transforming sequences of operation symbols into the corresponding traces. Such transformations can also be described in another formalism, namely that of term rewriting rules [HO].

We shall illustrate the derivation of rewrite rules by means of a simple example. Consider again trace (11). It denotes a state where customer B uses a machine, as can be seen from any of the preceding specifications: the informal, the semi-formal or the procedural one. Let us consider the problem of transforming the term uses(B, (11) ) into True.

We may start by trying to convert uses(B, (11) ) into simpler terms by moving uses inwards until the transformation is immediate. We are led to the following rules:

uses(x,lease(y,s)) → uses(x,s), whenever x ≠ y (16)

uses(x,sell(x,s)) → True (17)

These rules correspond to two possible paths in the execution of the procedure uses on the actual trace level, the first one corresponding to the path where t matches lease(y,s) and y is different from the customer parameter.

Notice that rule (16) has a precondition, which may be incorporated into the rule if we assume the Boolean sort equipped with an if-then-else, together with its natural specification. Then, (16) would be merged with its companion rule into:

uses(x,lease(y,s)) → if x = y then True else uses(x,s)

We shall not pursue here the example, as it is very similar to the equational one, treated in the next section. Some general remarks, however, are in order.

Consider a system R of rewrite rules and sets of terms $V,W \subseteq T$. We call V R-controllable to W, noted $V \xrightarrow{R} W$, iff for every $v \in V$ there exists $w \in W$ such that $v \xrightarrow{R} w$. Here we employ the usual notation $v \xrightarrow{R} w$ to denote that v can be rewritten as w according to the rules of R. Given an algebra A, call R sound on V with respect to A iff whenever $v \xrightarrow{R} t$ with $v \in V$ then $v^A = t^A$.

In order to specify a cta $C$ we need a system $R$ that is

($\alpha$) complete, in the sense $T \xrightarrow{R} C$, and
($\beta$) correct, in the sense that $R$ is sound on $T$ with respect to $C$.

Such a system $R$ must have the Church-Rosser property, but not necessarily that of finite termination, the role of the latter being played here by controllability.

Notice that each one of the procedural specifications presented for our running example is actually a deterministic implementation of a rewriting system with the above properties. In particular, on each level, $T$ is controllable to the corresponding canonical form. However, from the viewpoint of stepwise specification, there is a simpler alternative: we may relativize requirements ($\alpha$) and ($\beta$) to the preceding level. For instance, for the rewriting system of the reduced level, it suffices that

- the repetition-free canonical form be controllable to the reduced canonical form;
- the rules are sound on the repetition-free canonical form with respect to $\mathcal{D}$.

In general, our stepwise methodology for specifying a given algebra $A$ will consist of obtaining a sequence of trace algebras $T = C_0, C_1, \ldots, C_n \cong A$ with corresponding trace levels $\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_n$ and a sequence of rewriting systems $R_1, \ldots, R_n$ such that for each $k = 1, \ldots, n$

- $C_{k-1}$ is $R_k$-controllable to $C_k$;
- $R_k$ is sound on $C_{k-1}$ with respect to $\mathcal{C}_k$.

Theorem. Under the above conditions the rewriting system $R = R_1 \cup \ldots \cup R_n$ is a correct and complete specification for $C_n$, and $T/\equiv[R] \cong A$.

Furthermore, this approach leads naturally to a better documentation for $R$. Namely $R_1\{C_1\}\ R_2 \ldots R_{n-1}\{C_{n-1}\}\ R_n$, where the comment $\{C_k\}$ gives a description of the intermediate trace level. It has the advantage of suggesting a prospective user of the specification a good and safe way to use it; namely, first use the rules of $R_1$ to reduce into $C_1$, then apply the rules of $R_2$, etc.

The preceding general remarks refer to an arbitrary fi-

nitely generated algebra. In this case we have $C_0 = T$, $C_0$ being the actual trace. For the case of hierarchical data types, we have given a basic algebra $B$. We assume that the corresponding primitive congruence $=_p$ on the basic sorts to be given by means of a system $R_0$ of rewrite rules that is a correct and complete specification for the basic algebra $B$.

The relativization of our general methodology for the specification of the given hierarchical algebra A then is as follows

(1) Start with a rewrite system $R_0$, such that $\equiv[R_0]$ is consistent and sufficiently complete (with respect to the given primitive congruence).

(2) Obtain a sequence of hierarchical trace algebras $C_0, C_1, \ldots, C_n$ with corresponding hierarchical trace levels $C_0, C_1, \ldots, C_n$, so that $C_0$ is the actual hierarchical trace level and $C_n$ is a unique representative hierarchical trace level;

(3) Obtain a sequence of rewrite systems $R_1, \ldots, R_n$ for the terms of non-basic sorts, such that for $k = 1,\ldots,n$

- $R_k$ is sound on $C_{k-1}$ with respect to $C_k$.
- $C_{k-1}$ is $R_k$-controllable to $C_k$.

An important aspect of this stepwise methodology is its modularity with respect to sorts as well. In dealing with hierarchical data types we may assume the above $R_0$ as given, which amounts to assuming the basic algebra $B$ already specified. But, we can also back up and use the same general methodology to construct a specification for the basic algebra $B$ itself in a stepwise manner. This was illustrated in the beginning of this section when we obtained rewrite rules to convert uses(B, (11) ) into True.

## 9. Equational Specifications

In the previous section we tried to obtain rules to transform a term into the corresponding canonical form. These rules can be translated easily into conditional equations. Alternatively, we may ask ourselves what axioms would enable us to derive the equalities between terms and corresponding canonical forms. We shall illustrate this process within our stepwise approach, level by level.

### 9.1. Actual (hierarchical) trace level

Here, the equalities between terms of sort customer consist only of syntactical identities, no special equations being needed for them. All we need is a set of conditional equations allowing us to derive the correct answers for all queries.

The following 12 conditional equations, obtained by the reasoning outlined in the preceding section, will be:

uses(x,phi) = False (18)

owns(x,phi) = False (19)

uses(x,lease(x,s)) = True (20)

x ≠ y → uses(x,lease(y,s)) = uses(x,s) (21)

owns(x,lease(y,s)) = owns(x,s) (22)

uses(x,sell(x,s)) = True (23)

x ≠ y → uses(x,sell(y,s)) = uses(x,s) (24)

owns(x,sell(x,s)) = True (25)

x ≠ y → owns(x,sell(y,s)) = owns(x,s) (26)

uses(x,return(x,s)) = owns(x,s) (27)

x ≠ y → uses(x,return(y,s)) = uses(x,s) (28)

x ≠ y → owns(x,return(y,s)) = owns(x,s) (29)

Notice that these equations are arranged according to the leading operation symbol in the trace: phi, lease, sell, return and then according to the query: uses, owns.

### 9.2. Repetition-free (hierarchical) trace level

In section 6 we saw that (12) is a repetition-free trace corresponding to the actual trace (11). One way to derive the equality (11) = (12) is by means of axioms enabling the elimination

of the symbols of updates causing no net state change. This can be done by axioms (30) to (33) below, which should be added to the preceding ones to give an equational specification for the repetition-free trace level.

uses(x,s) = True → lease(x,s) = s (30)
owns(x,s) = True → sell(x,s) = s (31)
uses(x,s) = False → return(x,s) = s (32)
owns(x,s) = True → return(x,s) = s (33)

Notice that the last axiom above concerns violation of requirements, whereas the other three refer to redundant updates.

## 9.3. Reduced (hierarchical) trace level

Referring again to section 6, we see that (13) is a reduced trace corresponding to trace (12). In order to derive the equality (12) = (13) we need an equation like (34) below, which states that a return cancels an immediately preceding lease. In order to treat non-adjacent operation symbols we further introduce commutative axioms - both conditional ones like (40) and unconditional ones like (36), (37), (38) and (39).

return(x,lease(x,s)) = return(x,s) (34)
sell(x,lease(x,s)) = sell(x,s) (35)
lease(x,lease(y,s)) = lease(y,lease(x,s)) (36)
sell(x,sell(y,s)) = sell(y,sell(x,s)) (37)
lease(x,sell(y,s)) = sell(y,lease(x,s)) (38)
return(x,sell(y,s)) = sell(y,return(x,s)) (39)
x ≠ y → return(x,lease(y,s)) = lease(y,return(x,s)) (40)

Axioms (18) to (40) constitute an equational specification for this level.

## 9.4. Unique-representative (hierarchical) level

In our example, we can already derive from the previous level specification equalities like (13) = (14). In general, we may need some extra axioms, typically of commutativity, enabling the reordering of some terms.

Thus, conditional equations (18) to (40) constitute a correct and complete equational specification of our running example. We just remark that axioms (27), (28) and (29) are no longer necessary and may be discarded. Actually, these 3 axioms were no longer needed for the reduced trace level for the same reason: return no longer occurs in the traces.

## 10. Conclusion

The proposed methodology provides a multi-step strategy for the difficult task of obtaining an algebraic specification, noting that every step is within the algebraic formalism itself.

The methodology starts at a level where all ground terms are taken as representatives for states and gradually proceeds via a series of intermediate levels until reaching the desired level (say, that with a unique representative for each state). On the intermediate levels the specifications progress towards smaller sets of representatives by considering fewer sequences of updates as representatives. Typical (but not exhaustive) examples of criteria for this purpose are:

- not adding an update producing no net effect in a state;
- making a "negative" update to cancel the corresponding "positive" update;
- making an update whose effects subsume those of other to replace the latter;
- reordering some updates that commute.

Some general properties of these level specifications are worth mentioning:

- each level corresponds to a canonical term algebra;
- each level specification is sufficiently complete;
- the correctness criterion for each level is given by the observability relation '~': we have t~t' iff, for all queries q, $q(t,t_1,\ldots,t_n) = q(t', t_1,\ldots,t_n)$; i.e. t ~t' must imply that t and t' denote the same state;
- the set of trace levels is characterized in terms of lattices.

From an applied point of view, the intuitive meaning of traces as carrying a "history" of the data base deserves attention; the extra information available at the different trace levels may be of interest during the early experimental phase, made possible by the usage of executable specifications.

Finally we should stress that this methodology, theoretically proven correct, has been found quite useful in practice in the specification of a number of examples of data base applications.

References

[BP] Bartussek, W. and Parnas, D. - "Using traces to write abstract specifications for software modules" - technical report 77-012 - University of North Carolina (1977)

[EF] Ehrig, H. and Fey, W. - "Methodology for the specification of software systems: from formal requirements to algebraic design specifications" - Proc. GI-11. Jahrestagung - W. Brauer (ed.) - Springer (1981) 255-269.

[FV] Furtado, A.L. , and Veloso, P.A.S. - "Procedural specifications and implementations for abstract data types" - ACM/SIGPLAN Notices, vol. 16 no. 3 (1981) 53-62.

[FVC] Furtado, A.L., Veloso, P.A.S, and Castilho, J.M.V. - "Verification and testing of S-ER representations" - in 'Entity relationship approach to information modelling and analysis' - P.P.Chen (ed.) - E-R Institute (1981) 125-149.

[GH] Guttag, J.V. and Horning, J.H. - "The algebraic specification of abstract data types" - Acta Informatica, vol. 10, fasc. 1 (1978) 27-52.

[Gr] Grätzer, G. - "Universal algebra" - D. van Nostrand (1968).

[GTW] Goguen, J.A., Thatcher, J.W. and Wagner, E.G. - "An initial algebra approach to the specification, correctness and implementation of abstract data types" - in 'Current trends in programming methodology' - R.T. Yeh (ed.), vol IV - Prentice-Hall (1978) 80-149.

[HO] Huet, G. and Oppen, D.C. - "Equations and rewrite rules: a survey" - technical report STAN-CS-80-785 - Stanford University (1980).

[Li] Liskov, B.H. et al - "Abstraction mechanisms in CLU" - Comm. of the ACM, vol. 20, no 8 (1977) 564-576.

[Pa] Pair, C. - "Sur les modèles des types abstraits algébriques" - Séminaire d'Informatique Théoretique - Université de Paris VI et VII (1980).

[PV] Pequeno, T.H.C. and Veloso, P.A.S. - "Don't write more axioms than you have to" - Proc. International Computing Symposium 1978 (vol. 1) Academia Sinica (1978) 487-498.

[VCF] Veloso, P.A.S., Castilho, J.M.V. and Furtado, A.L. - "Systematic derivation of complementary specifications" - Proc. Seventh International Conference on Very Large Data Bases (1981) 409-421.

[Ve] Veloso, P.A.S. - "Methodical specification of abstract data types via rewriting systems" - International Journal of Computer and Information Sciences, vol. 11 nº 5 (1982) 295-323.

[WB] Wirsing, M. and Broy, M. - Abstract data types as lattices of finitely generated models" - Institut für Informatik - Tech. Univ. München (1980).