DEFINABLE OPERATIONS IN GENERAL ALGEBRAS, AND THE THEORY OF AUTOMATA AND FLOWCHARTS [1]

Hans Bekić, IBM Laboratory, Vienna

Abstract. We study the class of operations definable from the given operations of an algebra of sets by union, composition, and fixed points; we obtain two theorems on definable operations that give us as special case the regular-equals-recognisable theorem of generalised finite automata theory. Definable operations arise also as the operations computable by charts; by translating into predicate logic, we obtain Manna's formulas for termination and correctness of flowcharts.

To be published in Machine Intelligence 5, Edinburgh

Vienna, 8 December, 1969

---

1) Work carried out at Queen Mary College, London, under Research Grant B/SR/5987 of the S.R.C.

INTRODUCTION

The regular-equals-recognisable theorem of generalised (i.e. tree) automata theory can be viewed as asserting that the sets of terms, or trees, definable by equation systems in a certain normal form are the same as those obtainable from the empty set and unit sets by union, a kind of composition, and a kind of iteration. In the attempt to arrive at a better understanding of this theorem and its proof, it appeared that one could start from an arbitrary algebra rather than just from the algebra of terms. Given the operations of an algebra, it is usual to consider a larger set of operations, namely those "derivable" from the original ones by permutation of variables and composition. If we raise the given algebra to the set level (f applied to set = set of results of applying f to the elements), then we can use two other ways of extending the class of our operations: union, and (simultaneous) fixed points, i.e. definition by an equation system. It turns out that our original theorem can be gained as special case of the combination of two simple theorems on the class of "definable" operations so obtained. The first theorem asserts that in fact we can do with simple recursion. The second asserts that each definable operation can be defined by an equation system in normal form.

Conventional, i.e. string automata used to be represented by transition charts. Hence it is not surprising that normal operations - which where those giving us the recognisable sets - are precisely those computable by charts, also if the chart is not interpreted just in the algebra of strings (and also if we go to the generalised case, i.e. to arbitrary instead of unary algebras, generalising the notion of chart appropriately). Remaining for a moment at the unary case, i.e., at conventional flowcharts, we observe that the normal operation associated with the chart asks us to find the smallest system of sets satisfying certain mutual closure relationships. This is a typical second order problem, like e.g. the problem of defining the natural numbers, i.e. the smallest set containing 0 and closed under successor. It is second order, because, as we well know from the fifth Peano axiom, such definitions can be expressed in predicate logic only speaking about all properties, or all sets, of elements. Expressing our operation computed by a chart in predicate logic, we thus get second order formulas - or, since the formulas have only one kind of second order quantifier, validity or satisfiability of first order formulas; so here are Manna's conditions on termination and correctness of flowcharts.

It may be difficult enough to discover just a mechanism for telling precisely what a programming language means. It is still more difficult to find a formalisation that keeps describing a language, i.e. preserves the structural similarity between expressions and meanings. Expressions in conventional mathematics, like arithmetic expressions, can easily be defined as an algebra and their meaning as a homomorphism from the algebra of expressions to the algebra of numbers. Can we do something similar for charts? Conventional automata are charts, generalised automata have been defined as (finite) algebras. So it is only one step to an idea of P. Landin to define a chart

as a finite algebra (plus entries and exits). Executing the chart in an other algebra then means following equal paths in the two algebras; so the computation of the chart becomes the algebra generated in the direct product of the two algebras from the assignment of initial values.

In the first part of the paper (sections 1 to 3) we develop the theory of definable operations; instead of the set of subsets of a given set, partially ordered by set inclusion, we consider an arbitrary set with an arbitrary partial ordering. In the second part (sections 4 to 6) we return to algebras, automata, and flowcharts.

Notation. We use familiar notation from set theory, e.g. $\cup$ and $\bigcup$ for union, $\{\ \}$ or $\emptyset$ for the empty set. $\mathcal{B}A$ is the set of subsets of A; $A_1 X \ldots X A_n$ is the Cartesian product of the sets A_i, i.e. the set of n-tuples $(a_1,\ldots,a_n)$ with $a_i \in A_i$; A^n is the set of n-tuples of elements of A. A (total) function $f:A \to B$ is a mapping from A to B; $A \to B$ denotes the set of all such functions. A relation from A to B is a subset of $A \times B$. In the last section, we use propositional connectives and quantifiers from predicate logic.

1. DEFINABLE OPERATIONS IN COMPLETE LATTICES

In this section, we consider a set M partially ordered by a relation $\leqslant$, a set θ of operations on M with certain "good", i.e. order-preserving, properties, and certain functionals, i.e. operations on operations, that preserve goodness. As indicated, M in our applications will become $\mathcal{B}A$, the set of subsets of A, $\leqslant$ will become $\subseteq$, i.e. set inclusion, and θ will result from raising to the set level a given set of operations on A. One of our functionals is concerned with the construction of fixed points and is applicable only to good operations. Therefore we will start with discussing good functions and fixed points, and will then introduce our functionals and show that they preserve goodness. Our aim is to define a class $D\theta$ as the smallest class containing θ and closed under the functionals.

Some lattice-theoretic definitions (cf.e.g. Cohn [4] chapter I.4 and II.4). A set M is partially ordered by a binary relation $\leqslant$ on M, if $\leqslant$ is reflexive ($a \leqslant a$), transitive (if $a \leqslant b$ and $b \leqslant c$ then $a \leqslant c$), and antisymmetric (if $a \leqslant b$ then not $b \leqslant a$). For a subset L of M, $a \in M$ is the least upper bound (l.u.b.) of L, if a is an upper bound of L, i.e. $a \geqslant x$ for all $x \in L$, and if for every upper bound b of L we have $b \geqslant a$. (We write $b \geqslant a$ for $a \leqslant b$). Lower bound and greatest lower bound (g.l.b.) are defined similarly. We write $\bigvee L$, or, in the case $L=\{b,c\}$, $b \vee c$ for the l.u.b. of L; similarly $\bigwedge L$ or $b \wedge c$ for g.l.b.s; these notations are borrowed from the special case of $\mathcal{B}A$ ordered by $\subseteq$, where $\bigcup$ and $\bigcap$ become union and intersection. M is called a lattice, if $b \vee c$ and $b \wedge c$ exist for all $b,c \in M$; a complete lattice, if $\bigvee L$ and $\bigwedge L$ exist for all $L \subseteq M$.

Now let M be partially ordered by $\leqslant$. We will gradually introduce assumptions on the existence of l.u.b.s (assumptions 1,2,3 below) which together will be only slightly

weaker than postulating M as a complete lattice.

Good functions, minimal fixed points

From the theory of production systems and context-free languages we are familiar with definitions like (after translation into set notation) the following:

$$x = a \cup gxy$$
$$y = b \cup hjy$$

Here, a and b are sets of strings, gxy is the set of strings starting with the letter g, followed by a member of x followed by a member of y. This definition by "simultaneous recursion" is intended to describe x, y as the smallest sets x and y that satisfy the indicated equalities. Of course we know how to find x, y: Denoting by f the function which the pair $a \cup gxy$, $b \cup hxy$ is of x,y we start with the pair of empty sets: $x_0 = \emptyset$, $y_0 = \emptyset$, and iterate the process of applying $f: x_{i+1}$, $y_{i+1} = fx_i y_i$; then the x_i , y_i form an ascending chain, i.e. $x_i \subseteq x_{i+1}$, $y_i \subseteq y_{i+1}$, and the required pair x,y is the limit, or union, of the chain.

In the theorem below, we describe and justify this construction in the abstract. We start with two assumptions on M :

1. M has a minimal element 0, i.e. $0 \leqslant x$ for all $x \in M$.
2. Any countably infinite ascending chain $a_0 \leqslant a_1 \leqslant \ldots$ in M has a l.u.b. $\bigcup_i a_i$.

Definition: A function $f: M \to M$ is called good if it is monotone, i.e. $fa \leqslant fb$ for $a \leqslant b$, and continuous, i.e. $f \bigcup_i a_i = \bigcup_i fa_i$ for countable ascending chains a_i.

(In fact, monotonicity follows from continuity, but it is convenient to consider the two properties separately.) Note, that, by monotonoicity, the fa_i are ascending if the a_i are.

Theorem (Construction of minimal fixed points, "recursion theorem"). Let M be partially ordered with minimal element and l.u.b.s for countable ascending chains; let $f: M \to M$ be a good function. Then there is a minimal solution a of

$$x \geqslant fx$$

which is given by $a = \bigcup_i f^i 0$ and in fact satisfies a = fa, i.e. is a fixed point of f.

Proof. $0 \leqslant f0$, hence by monotonicity $f0 \leqslant f^2 0$, $f^2 0 \leqslant f^3 0, \ldots$, i.e. the $f^i 0$ are ascending. Now $fa = \bigcup_i f^{i+1} 0$ by continuity, i.e. $fa = a$. Further, $b \geqslant fb$ implies $b \geqslant fb$ implies $b \geqslant 0$, $b \geqslant fb \geqslant f0$, $b \geqslant fb \geqslant f^2 0, \ldots$ thus $b \geqslant \bigcup_i f^i 0$, i.e. $b \geqslant a$.

Notation. We call a the minimal fixed point of f and write $a = \mu x.fx$.

There are several well-known applications of this theorem. One is the recursion theorem in recursive function theory, where M is the set of partial recursive functions of natural numbers, partially ordered by "being an extension of "; the theorem justifies recursive definitions like the familiar one for the factorial function. Another application is the construction of sets closed under given operations, i.e. minimal subalgebras. This is the one we are interested in, and it will be discussed (using a slightly modified version of our theorem, see end of this section) in section 4.

If we were interested only in the existence of fixed points, not in the particular construction, we could use also the following theorem, which gets away with weaker assumptions on f (through stronger ones on M):

Theorem (Tarski's fixed point theorem, see Tarski [14]). Any monotone mapping of a complete lattice into itself has fixed points. In particular, $\bigcap\{x \mid x \geqslant fx\}$ is the minimal fixed point.

Note that the set of partial recursive functions in the previous example does not form a complete lattice.

Functionals preserving goodness

We add a third assumption on M:

3. M has l.u.b.s for pairs, i.e. if $a,b \in M$ then $a \cup b \in M$.

Thus we have now l.u.b.s for finite subsets (by 1 and 3; note that $0 = \bigcup\{\ \}$) and for countable ascending chains. By assuming l.u.b.s for every subset, we would in fact have a complete lattice (see e.g. Cohn [4] I.4).

Extending $\leqslant$ to tuples. For any $n \geqslant 0$, we consider the set M^n of n-tuples of elements of M and extend $\leqslant$ to a partial ordering of M^n defining

$$(a_1,\ldots,a_n) \leqslant (b_1,\ldots,b_n) \qquad \text{if } a_1 \leqslant b_1, \ldots, a_n \leqslant b_n$$

Thus $\leqslant$, and therefore $\bigcup$, are defined component-wise; conditions 1 to 3 extend to M^n, in particular, $(0,0,\ldots,0)$ is the minimal element.

Operations on M. Instead of functions $f: M \to M$, we want to consider more generally operations on M, i.e. functions $f:M^n \to M^p$ ($n \geq 0$, $p \geq 0$). We also write $f \in [n \to p]$ and say that f is of arity $n \to p$; sometimes we write $[n\to]$ for $\bigcup_p [n \to p]$, $[\to p]$ for $\bigcup_n [n \to p]$.

For $f:M^n \to M^p$ we define good component-wise, i.e. f is good if it is monotone and continuous under the ordering $\leqslant$ in m^n and M^p.

Notation for tuples. If $a = (a_1,\ldots,a_n)$, $b = (b_1,\ldots,b_m)$, we write (a,b) for $(a_1,\ldots,a_n,b_1,\ldots,b_m)$. (This notation is unambiguous, since we will only consider tuples of elements, not tuples of tuples). The 0-tuple is denoted by $()$. We will in general use unindexed letters a,b,...x,y for tuples, indexed letters for elements. In $(a_1,\ldots a_n)$ or (a,b), parantheses and commas may be omitted, in particular in the context $f(a_1,\ldots,a_n)$ or f(a,b).

The list of functionals. We now give our list of functionals, i.e. operations on operations on M. Actually, the list contains an element of M (namely 0), and operations on M (namely $e_i^n, 0^n$); these may be viewed as constant, i.e. 0ary, functionals. n, p etc. ≥ 0 unless otherwise stated.

1.a. components: $e_i^n \in [n \to 1]$; $e_i^n x_1 \ldots x_n = x_i$ $\quad (1 \leqslant i \leqslant n,\ n \geqslant 1)$

b. producing $()$: $0^n \in [n \to 0]$; $0^n x = ()$

c. tupling: for $f \in [n \to p]$, $g \in [n \to q]$:

$(f,g) \in [n \to p + q]$; $(f,g)x = (fx,gx)$

2.a. the minimal element 0 (considered as operation $\in [0 \to 1]$).

b. union: for $f,g \in [n \to p]$: $f \cup g \in [n \to p]$; $(f \cup g)x = fx \cup gx$

3. composition: for $f \in [m \to p]$, $g \in [n \to m]$:

$f \circ g \in [n \to p]$; $(f \circ g)x = f(gx)$

4. fixed points: for $f \in [n + p \to p]$, good: $f^\dagger \in [n \to p]$; $f^\dagger x = \mu y.fxy$

Thus 1 is concerned with tupling and untupling, 2 with finite unions, 3 with functional composition, 4 with fixed points, or definition by simultaneous recursion, relative to the last components (this is no restriction, since permutations are included by 2). 4 is justified by observing that M^p satisfies the assumptions of the recursion theorem and that, for given x, the function $g:M^p \to M^p$, $gy = fxy$, is good if f is good.

Lemma (preservation of goodness). Functionals 1 to 4, applied to good gunctions, yield good functions.

Proof (Note: for 0ary functions, i.e. operations, this means that the operation is good.)

1.abc : Goodness is defined component-wise.

2.a. : any constant (i.e. 0ary operation) is good.

b. : if $fb \geqslant fa$, $gb \geqslant ga$ then $fb \cup gb \geqslant fa \cup ga$;

also, $\bigcup_i (fa_i \cup ga_i) = \bigcup_i fa_i \cup \bigcup_i ga_i$.

3. If $b \geqslant a$ then $fb \geqslant fa$, hence $g(fb) \geqslant g(fa)$;

$\bigcup_i g(fa_i) = g \bigcup_i fa_i = g(f \bigcup_i a_i)$.

4. For $b \geqslant a$, we have $fby \geqslant fay$, i.e. each $y \geqslant fby$ is also $\geqslant fay$, thus $\mu y.fay \geqslant \mu y.fby$. To prove continuity we put $a = \bigcup_i a_i$, $y_i = \mu y.fa_i y$ and have to show $\mu y.fay = \bigcup_i y_i$. Now for $y \geqslant fay$, we have $y \geqslant fa_i y \geqslant y_i$, thus $y \geqslant \bigcup_i y_i$; conversely, $y = \bigcup_i y_i$ satisfies $y \geqslant y_i = fa_i y_i$, thus $y \geqslant \bigcup_i fa_i = fay$.

The set Dθ

Definition. Let a set θ of good operations over M, i.e. good functions $\in \bigcup_{n,p} M^n \to M^p$ be given. The set Dθ of operations definable from θ (or of definable operations, if θ is understood), is the smallest set containing θ and closed under functionals 1-4.

This definition is justified by the previous lemma: since we never leave the set of good operations, $\dagger$ is always applicable. By closed under a given function, e.g. $\circ$, we mean that $f \circ g$ is in the set whenever f and g are in the set and satisfy the conditions on their arities as stated in the definition of $\circ$. In particular, for a 0ary functional we mean that the operation denoted by it is in the set. Thus we could also say that Dθ is the smallest set containing θ and the operations 1a, 1b and 2a and closed under 1c, 2b, 3 and 4.

where-where rec-notation

In discussing definable operations, we will often find it more convenient to use a notation that provides for variables and auxiliary definitions, rather than to use explicitly the functionals introduced in 1-4. If E and F are expressions denoting after assignment of elements or tuples of elements of M to their free variables,

elements or tuples of elements of M, then, denoting by $\lambda x.E$ the function which E is of x, we will write

E where x = F	for	$(\lambda x.E)F$
rec x = F	for	$x = \mu x.F$

Thus E where rec x = F for E where rec x = μx.F, i.e. for $(\lambda x.E)\ (\mu x.F)$. Note that a x which occurs free in F is not bound by the where-clause in E where x = F, but is bound by the where-rec-clause in E where rec x = F. Thus our use of where corresponds to that formalised in Landin [7] rather than to that in most of informal mathematics (which is nearer to where rec). Again our use of rec is that introduced in [7]: there rec x = F was explained as Y $(\lambda x.F)$, where Y is a fixed point finder; we are lucky to be in the special situation where we know precisely what Y is.

Translating from functional notation to where - where rec-notation

Our expressions in where-where rec-notation will be built up from 0 and variables by tupling, symbols denoting operations in θ, and auxiliary definitions. Given an operation $h \in D\theta$ by a derivation using functionals 1-4, we can find an expression whose value, as a function of its free variable x, is hx by using the rules $(f\circ g)x = f(gx)$ and $g^{\dagger}x = (y$ where rec y = gxy), together with obvious rules for $(f,g)x$, $0^n x$, $e^n_i x_1 \ldots x_n$ (other uses of e^n_i can be reduced to this one by introducing auxiliary definitions).

Translating from where - where rec-notation to functional notation

To find,
conversely, a derivation for the function which a given expression is of x, we can use (again together with a few more trivial rules)

$$(fxy \text{ where } y = gx) = f(x,gx) = (f\circ(I,g))x$$

where I is the identity on x , i.e. $I = (e^n_1,\ldots e^n_n)$ if y ranges over M^n, and

$$(fxy \text{ where rec } y = gxy) = (fxy \text{ where } y = g^{\dagger}x)$$

where the right hand side is to be transformed further by the first rule.

Example. Suppose $a,f \in \theta$. The (0ary) operation $\mu y.a\cup hay$ is written y where rec y = $a\cup hay$ in where - where rec-notation , $[a\circ 0^1 \cup h\circ (a\circ 0^1 , e^1_1)]^{\dagger}$ in functional notation.

Cumulative iteration vs. fixed points

The minimal fixed point $\mu x.fx$ was defined as $\bigcup_i f^i 0$. This suggests the question whether we could have used iteration, i.e. the functional * defined by $f^*x = \bigcup_i f^i x$, instead of μ (and hence iteration with respect to the last components instead of $\dagger$) in building up D0. The answer to this question should be no (even under the necessary existence assumptions on $\bigcup_i$), but we will show that we can use cumulative iteration $\cup^*$ defined by $f^{\cup^*}x = \bigcup_i (I \cup f)^i x$, where I is the identity function on the domain of f, i.e. $I = (e_1,\ldots,e_n)$ for $f \in [n \to]$. We note $(I \cup f)x = x \cup fx \geqslant x$, i.e. the $(I \cup f)^i x$ are ascending (whereas the $f^i x$ in general are not). The following lemma expresses $\cup^*$ in terms of μ (and in fact gives the recursion theorem for $x = 0$).

Lemma. For M satisfying the assumptions of the recursion theorem, and good $f:M \to M$:

$$f^{\cup^*}x = \mu y.x \cup fy$$

Proof. Let $a = f^{\cup^*}x$. We have $a \geqslant x$ and also $a = (I \cup f)a \geqslant fa$, thus $a \geqslant x \cup fa$. For $b \geqslant x \cup fb$, we have $b \geqslant x$, $b \geqslant (I \cup f)x$, $b \geqslant (I \cup f)b \geqslant (I \cup f)^2 x \ldots$, thus $b \geqslant \bigcup_i (I \cup f)^i x = a$.

Conversely, $\mu x.fx = \bigcup_i f^i 0 = \bigcup_i (I \cup f)^i 0 = f^{\cup^*}0$, i.e. μ is expressible in terms of $\cup^*$. Thus $\cup^*$ would be as good as μ in building up D0; this result will be used in section 5 for comparing two definitions of regular set.

2. THEOREM I: ELIMINATION OF SIMULTANEOUS RECURSION

We have included in our list of functionals the use of μ with respect to several variables, because it is convenient to have an operator solving simultaneous equation systems, e.g. in the definition of recognisable set in automata theory. We will now prove that we could have done as well with μ on single (element) variables, hence that simulaneous recursion $\dagger$ can be defined by simple recursion $\dagger_1$ i.e. by the restriction of $\dagger$ to operations in $[n+1 \to 1]$.

Given the equation system $x = fxy$, $y = gxy$, we can first solve the equation for y (with x as parameter), then substitute the solution in the equation for x and solve it for x. This is expressed by the following lemma:

Bisection lemma. For good f,g $[p+q\ p]$, $[p+q\to q]$, respectively:

$$\mu xy.(fxy,gxy) = x_0,y_0$$

where

$$x_0 = \mu x.f(x,\mu y.gxy)$$
$$y_0 = \mu y.gx_0y$$

(with certain tacit assumptions on the ranges of x and y.)

Proof. We have $x_0 = \mu x.f(x,g^\dagger x)$, $y_0 = g^\dagger x_0$. Now $y_0 = gx_0y_0$, $x_0 = f(x_0,g^\dagger x_0) = fx_0y_0$, i.e. x_0,y_0 is a fixed point of (f,g). Conversely, if $x,y \geqslant fxy, gxy$, then $y \geqslant g^\dagger x$, $x \geqslant f(x,g^\dagger x)$, thus $x \geqslant x_0, y \geqslant g^\dagger x_0 = y_0$; hence x_0,y_0 is the minimal fixed point.

In where - where rec-notation, the lemma reads

$$(x,y \text{ where rec } x,y = fxy,\ gxy) =$$
$$= x,\ (y \text{ where rec } y = gxy) \text{ where rec } x = f(x,(y \text{ where rec } y = gxy)).$$

We note the special $fxy = hx$:

$$(x,y \text{ where rec } x,y = hx,gxy) =$$
$$= x,\ (y \text{ where rec } y = gxy) \quad \text{where rec } x = hx$$

Theorem I (elimination of simultaneous recursion). † can be defined in terms of $^{\dagger 1}$ and the remaining functionals.

Proof. For $f \in [\to 0]$, i.e. $fx = ()$, we have $f^\dagger x = ()$, i.e. $f^\dagger = f$. For $h \in [\to p+q]$, there are $f \in [\to p]$, $g \in [\to q]$ such that $h = (f,g)$ (and f and g are in $D\theta$ if h is). It follows from the lemma that $h^\dagger$ can be expressed in terms of $f^\dagger$ and $g^\dagger$ (the actual expansion is

$$(f,g)^\dagger = (f \mathbin{\acute{\circ}} g^\dagger)^\dagger\ ,\ g^\dagger \mathbin{\acute{\circ}} (f \mathbin{\acute{\circ}} g^\dagger)^\dagger$$

where $f \mathbin{\acute{\circ}} g$ is $f \circ (I,g)$ with suitable identity function I). Thus † on $\bigcup_{p>1} [\to p]$ can be gradually reduced to $^{\dagger 1}$.

3. THEOREM II : NORMAL FORM

In this section we define a normal form for operations on M and show that each operation in $D\theta$ is representable in normal form. Operations in normal form arise (in the case $m = BA$) as the operations computed by charts, and we will introduce and use charts

informally below for illustration. (A formal treatment of chart and operation computed by a chart is given in section 4). Our proof of the normal form theorem can be viewed as introducing operations on representations in normal form, or charts, corresponding to the functionals 1-4 on operations and as showing that the mapping "operation computed by a chart" is a homomorphism from the algebra of charts to the algebra of operations.

Assumption. We assume $\theta \subseteq [\to 1]$, i.e. the $f \in \theta$ produce elements rather than tuples. (This assumption is for convenience only; without it we had to base our definition of normal form on components of $f \in \theta$ rather than on $f \in \theta$.)

Normal form

Consider the following definition of a (0ary) operation (the example at the beginning of section 2, with changed names of variables, written in where - where rec-notation with the licence of splitting up definitions of tuples into definitions of their components):

$$y_1, y_2 \text{ where rec } y_1 = a \cup g y_1 y_2 \quad y_2 = b \cup h y_1 y_2 \qquad \text{(Fig. 1)}$$

We have associated a chart with it by generalising the usual notion of chart (e.g. transition chart in conventional automata theory): we allow "polyarcs", like the ones labelled g and h in the example, i.e. arcs leading from tuples of nodes to nodes, rather than from nodes to nodes. (In fact also the nullaries a and b give rise to polyarcs, namely to ones joining a 0-tuple of nodes to a node).

In this example (and in anyone of a 0ary operation for which there is a similar chart), the right hand side of each auxiliary definition consists of alternatives each of which is an operation $\in \theta$ applied to a tuple of variables (corresponding to an arc in the chart); the main clause is a tuple of variables (corresponding to a tuple of nodes marked as "exits" in the chart).

More generally, for an operation $\in [n \to p]$, we need a tuple $x_1, \ldots x_n$ of argument variables (corresponding to a tuple of nodes marked as "entries" in the chart):

$$y_1, x_2 \text{ where rec } y_1 = g x_1 y_2 \quad y_2 = x_2 \cup f y_1 \qquad \text{(Fig. 2)}$$

Like the y_j we allow the x_i as components of the main part (exits) and as arguments to $f \in \theta$. Unlike the y_j, we have to allow the x_i as alternatives on the r.h.s. (giv-

giving rise to an unlabelled arc); otherwise we could not even define e_1^1. Another difference is that we want the x_i to be free, i.e. not to appear as the left-hand side of a definition (thus no arc leading into them), whereas the y_j are bound (if only by a definition with no alternatives, i.e. 0 on the r.h.s.). (We treat the x_i as <u>variables</u> translating into <u>nodes</u>, rather than as (variable) <u>nullaries</u> translating into (nullary) <u>arcs</u>; in our formal treatment of charts we will choose the latter alternative, whereas here we prefer the slightly greater freedom of the former.)

To give now our definition of normal form, we call an operation on M <u>degree 0</u>, if it can be obtained by functionals 1 a,b,c alone, i.e. is a permutation with gaps and repetitions; <u>degree 1</u>, if it is $f \circ e$, where $f \in \theta$ and e is degree 0.

<u>Definition</u>. An operation f on M is <u>normal</u> if it can be represented in <u>normal form</u>, i.e. in the form

$$fx = e(x, \mu y.dxy)$$

(thus $f = e \circ (I, d^\dagger)$), where e is degree 0, $d = d_1, d_2, \ldots, d_m$), $d_j \in [\to 1]$, with each d_j a finite union of operations each of which is either degree 1 or is degree 0 in x, i.e. $e'xy = e'x$ with e" degree 0 (thus e" is some e_i^n).

<u>The normal form theorem</u>

Clearly each normal operation is in $D\theta$. We want to show the converse.

<u>Theorem II</u>. (<u>normal form theorem</u>). Each operation in $D\theta$ is normal

<u>Normal form with spontaneous moves</u>. Our proof of the theorem will produce as intermediate results representations in <u>normal form with spontaneous moves</u>; these are defined like representations in normal form, except that the e' may now be <u>arbitrary</u> degree 0 operations. They correspond to allowing the y_j as alternatives on the r.h.s. of a definition (thus to allowing "spontaneous moves" i.e. unlabelled arcs from y_j, in the chart). Spontaneous moves are familiar from conventional automata theory. Perhaps we should use the term more properly for <u>any</u> unlabelled arc; however we needed a name for the restricted case; for nullary operations i.e. when no x_i are present, the two uses coincide.

<u>Some identities</u>. We prove some simple identities on <u>where - where rec</u>-expressions, which we will use without explicit reference.

1. $(y \text{ where rec } y = a) = a$

Proof: the l.h.s. satisfies $y = a$ by definition.

2. (distribution:) $f(x \text{ where rec } x = gx) = fx \text{ where rec } x = gx)$

Proof: by definition of where, $f(x \text{ where } x = a) = fa = (fx \text{ where } x = a)$;
put $a = \mu x.gx$.

3. ("moving right":) $(fx \text{ where rec } x = gx) \text{ where rec } y = hy$
$= fx \text{ where rec } x,y = gxy,hy$

Proof: From the bisection lemma (special case), we have

$(x,y \text{ where rec } x,y = gxy,hy) =$
$= (x \text{ where rec } x = gxy),y \text{ where rec } y = hy$

Apply to both sides $\lambda xy.fx$, using 2.

4. ("moving left":) $(fx \text{ where rec } x = (gxy \text{ where rec } y = hxy)) =$
$= (fx \text{ where rec } x,y = gxy, hxy)$

Proof: similar.

Note. 3. and 4. (as well as the bisection lemma) become intuitively obvious if one remembers that the "scope" of a where rec-definition is the expression qualified by the definition, plus the definiens.

Proof of theorem II (up to spontaneous moves).

We show that $f \in \theta$ is normal, and that each of the functionals 1-4, when applied to operations in normal form, yields an operation representable in normal form with spontaneous moves. The corresponding operations on charts are very simple, e.g. joining exits of one chart with the corresponding entrances of the other for composition, with the last entrances of itself for †. To complete the proof of the theorem, we will show how eliminate spontaneous moves.

0. For $f \quad \theta$, we have $fx=(y \text{ where rec } y = fx)$. (Note $\theta \ [\to 1]$, i.e. $f \quad [\to 1]$).

1a. $e_i^n x_1 \ldots x_n = x_i$ (Note that in $e(x, \mu y.dxy)$, y may be (), i.e. we may have no auxiliary definitions at all).

1b. $0^n x = ()$

1c. (exy where rec y = dxy, e'xz where rec z = d'xz)
= (exy, e'xz) where rec y,z = dxy, d'xz
= (e∘c, e'∘c')xyz where rec y,z = (d∘c, d'∘c')xyz

where cxyz = xy, c'xyz = xz. If e and e' are degree 0, then so are e∘c, e'∘c'; if the components of d and d' are unions of operations which are degree 1, or degree 0 in x, then so are the components of (d∘c, d'∘c'). (In other words, dxy etc. continue to satisfy the assumptions if viewed as functions of more variables).

2a. 0 = (y where rec y = 0)

2b. (exy where rec y = dxy) ∪ (e'xz where rec z = d'xz)
= exy ∪ e'xz where rec y,z = dxy, d'xz
= u where rec u,y,z = exy ∪ e'xz, dxy, d'xz

(In this and the following two cases, we may get spontaneous moves).

3. (exy where rec y = dxy) where x = (e'x'z where rec z = d'x'z)
= exy where rec y,x,z = dxy, e'x'z, d'x'z

4. z where rec z = (exzy where rec y = dxzy)
= z where rec z,y = exzy, dxzy

(This gives us a representation for $\mu z.fxz$, i.e. for $f^{\dagger}x$, if we have one for fxz).

Elimination of spontaneous moves

Assume we have a representation of f in normal form with spontaneous moves:

$$fx = exy \text{ where rec } y = dxy$$

Choose some index k. To eliminate spontaneous moves from y_k, i.e. alternatives y_k in the $d_j xy$, proceed as follows: For each j, decompose the equation $y_j = d_j xy$ into

$$y_j = d'_j xy \cup \delta_j y_k$$

where $\delta_j z$ is either z or 0, and $d'_j xy$ does not contain an alternative y_k, and replace it by the equation

$$y_j = d'_j xy \quad \text{... for } j = k$$
$$y_j = d'_j xy \cup \delta_j (d'_k xy) \quad \text{... for } j \neq k$$

Thus we delete an alternative y_k in the equation for y_k, and substitute the modified r.h.s. for alternatives y_k in the other equation, thereby eliminating all alternatives y_k. Clearly, $z \geqslant gz \cup z$ is equivalent to $z \geqslant gz$; therefore the modification to the equation for y_k does not change the set of solutions of the system of inequations, in particular it preserves the minimal solution (of the inequations, hence of the equations). Modifying now the other equations produces an equivalent system even of equations, since we substitute according to an equation which remains in the system.

4. ALGEBRAS, CHARTS, THE OPERATION COMPUTED BY A CHART

We proceed now to the application of our results to the theory of automata and flowcharts and specialise our system M,θ to $\mathcal{B}A$, the set of subsets of a given set A, together with a set of fully additive operations on $\mathcal{B}A$, i.e. to a raised algebra (see below). We find it convenient to present first a generalised notion of chart (corresponding, with small variations, to the charts used informally in the previous section) and of the operation computed by a chart; these notions will cover the conventional flowchart and the transformation denoted by it, but also the (generalised) automaton and the set recognised by it. Our formalisation of charts is essentially due to P. Landin (see Landin [8]): it defines a chart as a finite (partial, nondeterminate) algebra plus entries and exits, and considers the computation of the chart in a second algebra as the process of generating pairs of elements with corresponding derivations, i.e. as generating the smallest subalgebra of the Cartesian product of the two algebras that contains the pairing of the entries with given initial values.

We start with a short survey of the necessary concepts from algebra. For a fuller treatment we refer the reader to Cohn [4] and in particular to Burstall and Landin [3] which contains many examples from programming; specifically for partial, nondeterminate algebras (usually called relational algebras) also to Grätzer [6] and Eilenberg-Wright [5].

Nondeterminate functions

By a nondeterminate function f from A to B, we mean a mapping $f: A \to \mathcal{B}B$, i.e. equivalently, a relation $f \subseteq A \times B$. We write fa for $\bigcup_{x \in a} fx$. By a nondeterminate element of A, we mean a subset of A, except that we include elements, i.e. identify elements with unit sets.

Basic definitions from algebra

Partial nondeterminate algebras. Let Ω be a set of operators with arity, i.e. Ω is the union of disjoint sets Ω_n, $n \geqslant 0$; we say ω has arity n if $\omega \in \Omega_n$. (More generally we consider arities $n \to p$; our restriction, which is the usual one, corresponds to the

restriction on θ from the beginning of last section).

A partial nondeterminate Ω-algebra (for short: an Ω-algebra) is a set A together with a mapping that associates with each $\omega \in \Omega_n$ a nondeterminate function ω_A from A^n to A. The set A is called the carrier of A; we usually do not distinguish in our notation between the algebra and its carrier. Also we usually write ω for ω_A, if the underlying algebra is understood. We speak of a total, determinate algebra if the ω_A are (total, determinate) functions $\in A^n \to A$.

Raising. The raised algebra $\hat{A}$ is the (total, determinate) algebra with carrier BA defined by

$$\omega a_1 \ldots a_n = \bigcup_{x_i \in a_i} \omega x_1 \ldots x_n \qquad (\omega \in \Omega_n)$$

(i.e. $\omega a = \omega(\Pi a)$, where Πa is the Cartesian product of the tuple of sets a).

$\hat{A}$ as a good system: BA, ordered by $\subseteq$, is a complete lattice. The $\omega_{\hat{A}}$ are completely additive, i.e. $\omega(\bigcup a) = \bigcup(\omega a)$, hence monotone and continuous. Thus βA, $\{\omega_{\hat{A}} \mid \omega \in \Omega\}$ satisfy the assumptions an M,θ in section 1.

Subalgebras. $A' \subseteq A$ is a subalgebra of A if $\omega_{A'} : A'^n \to B(A')$ is the restriction to A'^n of $\omega_A : A^n \to BA$, i.e. $\omega_{A'} a = \omega_A a$ for all $a \in A'^n$.

Therefore a subset $A' \subseteq A$ can be made into a subalgebra (and then uniquely so), if and only if A' admits the $\omega \in \Omega$, i.e. $\omega_A a \subseteq A'$ for all $a \in A^n$. The subalgebra generated from a subset $x \subseteq A$ is the smallest subalgebra containing x.

The existence of this subalgebra follows from the fact that subalgebras form a complete lattice. Existence and particular construction also follow from our lemma at the end of sectionl if we take for $f : BA \to BA$ the function $fy = \bigcup_n \bigcup_{\omega \in \Omega_n} \omega y^n$. (We would have to prove yet that f is good; in our applications, Ω will be finite and hence f definable, thus good).

The minimal algebra of A is the algebra generated from the empty set.

Direct product. The direct product A×B of algebras A and B is the algebra whose carrier is the Cartesian product A×B of the carriers and whose operations are

$$\omega_{A\times B}(a,b) = \omega_A a \times \omega_B b$$

Homomorphism, isomorphism. For these definitions we consider only total, determinate algebras, though a more general definition would be possible. Thus for total, determinate algebras A and B : A mapping $f : A \to B$ is a homomorphism from A to B if $\omega(fa_1 \ldots fa_n) = f(\omega a_1 \ldots a_n)$, i.e. defining $f(a_1, \ldots a_n)$ as $fa_1, \ldots fa_n$:

$$\omega \circ f = f \circ \omega \ .$$

An isomorphism is a homomorphism that is one to one and onto.

The free algebra. The total, determinate algebra T is called the free Ω-algebra if for every total, determinate Ω-algebra A there is a unique homomorphism from T to A.

We say the free algebra because any two of them are isomorphic. To see that there is a free algebra, we take from T the algebra of terms: $\omega \in \Omega_0$ is a term; if $t_1, \ldots t_n$ are terms and $\omega \in \Omega_n$ $(n \geqslant 1)$, then $\omega t_1 \ldots t_n$ is a term. We define $\omega_T(t_1, \ldots t_n) = \omega t_1 \ldots t_n$. (More formally, we take the set of tuples over Ω, define it as Ω-algebra by $\omega t_1 \ldots t_n = (\omega, t_1, \ldots, t_n)$ and define T as the minimal subalgebra.)

Charts

We assume from now on that Ω is finite. Given a finite Ω-algebra B, we can draw a chart like the ones used in section 3 (without entries and exits) as follows: Take as nodes the elements of the algebra. If $b \in fb_1 \ldots b_n$ in the algebra, draw a polyarc, labelled with the operator f, that joins the n-tuple of nodes $b_1, \ldots b_n$ with the node b. Conversely, given the chart, we can interpret its nodes as the elements of an algebra and the polyarcs as describing the extension of the operations of the algebra. Providing for entries and exits, we define therefore:

Definition. A chart is a finite Ω-algebra B, together with a n-tuple r of nondeterminate elements of B called entries, and a p-tuple s of nondeterminate elements of B called exits.

Example: In the following chart, there are four elements $b_1, \ldots, b_4$; there is one operation f defined by $fb_2b_1b_3 = b_4$, $fx_1x_2x_3 = \emptyset$ otherwise (there may be other operators in Ω, but in this algebra they would produce everywhere $\emptyset$) ;

(Fig.3)

r is $(\{b_1, b_3\}, b_2)$; s is b_4. (In drawing the chart, we have numbered the entrances and exits; similarly a numbering of components of polyarcs, or else some suitable convention, would be necessary).

Note that there are some differences to our informal charts concerning entries and exits, for example we did not provide there (and in our definition of normal form) for multiple exits.

The operation computed by a chart

Given a chart (B,r,s), we want to interpret it in a second algebra A. Like of B we may also think of A as given by a set of nodes connected by polyarcs describing the graphs of the operations. (This set will in general be infinite). Given a n-tuple x of initial values in A, we pair these with the corresponding entries in r; for nondeterminate x_i or r_i, we pair each element of the one with each of the other. We follow then polyarcs with equal labels to get further pairings and are interested in the p-tuple of sets of those elements of A that can in this way be paired with the corresponding exits in s. This suggests the following definition:

Definition. The operation computed by the chart (B,r,s) in the algebra $\hat{A}$ is the function $f : (BA)^n \to (BA)^p$ given by

$$fa = R_a s$$

where R_a is the subalgebra generated from r×a in B×A. (Remember that we can consider $R_a \subseteq B\times A$ as $R_a : B \to BA$.)

Notation. For fixed Ω, we take some fixed set $\{x_1, x_2, ..\}$ of nullaries disjoint from Ω. Given an Ω-algebra A and an n-tuple $(a_1,...a_n)$ of nondeterminate elements of A, we write $A[a_1,...,a_n]$ for the $(\Omega \cup \{x_1,...,x_n\})$ - algebra A' with carrier A and operations

$$\omega_{A'} = \omega_A \quad \text{for} \quad \omega \in \Omega$$
$$\omega_{A'} = a_i \quad \text{for} \quad \omega = x_i$$

The subalgebra R_a can then also be described as the minimal subalgebra of (B×A)[r×a], i.e. of B[r] × A[a].

Lemma. Let P be the minimal subalgebra of B×A. The sets Pb, for $b \in B$, are the smallest sets satisfying the conditions

$$\text{if } b \in \omega b_1 ... b_n \text{ then } Pb \supseteq \omega(Pb_1,...,Pb_n)$$

for all $n \geq 0$, $\omega \in \Omega_n$, $b, b_1,...,b_n \in B$.

Proof. As minimal subalgebra, P satisfies $P \supseteq \omega P^n$. This is equivalent to the condition in the lemma: both conditions express

$$\underline{\text{if}} \quad b \in \omega b_1 \ldots b_n,\ a \in \omega a_1 \ldots a_n,\ (b_i, a_i) \in P \quad \underline{\text{then}} \quad (b,a) \in P.$$

Also, for $P \subsetneq P'$ the sets Pb, P'b satisfy $Pb \subseteq P'b$ and conversely. Hence the minimal solutions coincide.

In our charts, we can view the entries as additional nullaries. This suggests the following slightly modified definition of normal operation on $\hat{A}$: An operation f on $\hat{A}$ is normal-1 if fa is a normal nullary on $\hat{A}[a]$ for all $a \in \hat{A}^n$. Clearly, for nullary f our two definitions of normal operation coincide. It is not difficult to show that they coincide for arbitrary f (but we will not use this result). We get now as corollary to the lemma our characterisation of the operations computable by charts:

Corollary: An operation on $\hat{A}$ is computable by a chart if and only if it is normal-1.

Proof. We use the lemma with B[r] × A[x]. A normal-1 operation f has the form fx= e(μy.dxy). Given such an f, we can immediately find a finite B[r] (by "reading" y = dxy as an algebra) such that, by the lemma, the $R_x b$ are the minimal solutions of y = dxy; ey gives us the p-tuple of exits. Conversely, given (B,r,s), we find by the lemma an equation system y = dxy for the $R_x b$. Then the operation f is

$$fx = (uy \ \underline{\text{where rec}}\ y = dxy)$$

where the components of uy are unions of components of y. Now

$$\begin{aligned} fx &= (z \ \underline{\text{where rec}}\ z = uy,\ y = dxy) \\ &= (z \ \underline{\text{where rec}}\ z = u(dxy),\ y = dxy)\ . \end{aligned}$$

The components of u(dxy) are unions of components of dxy and hence satisfy the conditions on normal form. (Here we have done some simple spontaneous move elimination "by hand").

5. AUTOMATA : KLEENE'S THEOREM

The theory of generalised automata arose from the observation that a conventional automaton, i.e. string automaton, can be regarded as a special algebra (one operator nullary, the rest unary), and the suggestion to extend the study to arbitrary algebras.

Definition (1). An automaton is a finite algebra B, together with a subset b_F.

Definition (2). An automaton is a chart $(B, (), b_F)$ with no entries and one (nondeterminate) exit.

Let val : $T \to \hat{B}$ be the unique homomorphism from the free algebra T to $\hat{B}$. If we view val as a relation, val $\subseteq T \times B$, it is easy to prove (by induction on T) that val is the minimal subalgebra of $T \times B$; therefore $val^{-1} \subseteq B \times T$ is the minimal subalgebra of $B \times T$.

Definition (1). The set recognised by an automaton (B, b_F) is the set of terms $t \in T$ for which $val \in t\ b_F$, i.e. the set $val^{-1}\ b_F$.

Definition (2). The set recognised by an automaton $(B, (), b_F)$ is the $(0 \to 1)$ ary operation on, i.e. subset of, $\hat{T}$ computed by the chart $(B, ()b_F)$ in $\hat{T}$.

According to the preceding characterisation of val^{-1}, the two definitions are equivalent.

Definition. A subset of T is recognisable if it is recognised by some automaton.

It follows that the recognisable subsets of T coincide with the $(0 \to 1)$ ary operations on $\hat{T}$ computable by a chart, hence, by the result of last section, with the $(0 \to 1)$ ary normal-1, i.e. the $(0 \to 1)$ ary normal, operations on $\hat{T}$.

Call an operation definable$_1$ if it is definable under our list of functionals 1-4, but with † replaced by $^{\dagger 1}$ (see section 2).

Definition. A subset of T is regular if it is a $(0 \to 1)$ ary definable$_1$ operation on T.

Now definable$_1$ = definable (theorem I), definable = normal (theorem II), hence in particular definable$_1$ $(0 \to 1)$ ary = normal $(0 \to 1)$ ary, thus:

Theorem (Kleene-Thatcher-Wright). The regular and recognisable subsets of T coincide.

Thatcher-Wright's definition of regular set. In Thatcher-Wright [15], regular sets are defined as the sets that are Ω'-regular for some $\Omega' = \Omega \cup \{x_1, \ldots, x_n\}$, where the x_i are additional nullaries. A subset of the free Ω'-algebra $T_{\Omega'}$, is Ω'-regular if it belongs to the smallest system containing all the finite subsets and closed under $\cup$, $\overset{\circ}{\varepsilon}$, $^{\varepsilon}$ (for all $\varepsilon \in \Omega'_o$), where

$u \underset{\varepsilon}{\circ} v$ = set of all terms obtainable from terms $t \in u$ by replacing all occurrences of ε in t by terms in v,

$$u^{\varepsilon} = \bigcup_i \{\lambda y.y \cup u \underset{\varepsilon}{\circ} y\}^i \{\varepsilon\}.$$

To relate this definition to ours, we associate with each Ω'-regular set u an expression E such that the value of E (interpreted in $\hat{T}$) under the assignment of sets $v_1,\ldots,v_n$ to variables $x_1,\ldots,x_n$ is $u\,x_1,\ldots\overset{\circ}{,}x_n\,(v_1,\ldots,v_n)$, where $x_1,\ldots\overset{\circ}{,}x_n$ is simultaneous substitution. It is clear how to deal with $\emptyset$, $\{x_i\}$, $\{fx_1\ldots x_n\}$ (for $f \in \Omega_n$), and $u \cup v$. For $\underset{y}{\circ}$ and ^{y} we use the translation

$$u \underset{y}{\circ} v \qquad u \underline{\underline{\text{where}}}\ y = v$$

$$u^{y} \qquad (\mu y.z \cup x) \ \underline{\underline{\text{where}}}\ z = y$$

(see the lemma at the end of section 1). The resulting expression denotes a definable$_1$ operation of its free variables. Conversely, given such an operation, we may assume that we have a $\underline{\underline{\text{where - where rec}}}$ - expression for it that has variables for elements only, no simultaneous definitions, and no composite terms (like $f(gx)$). Then we get the corresponding set by the same rules, except that for $\underline{\underline{\text{where rec}}}$ we translate into μ-notation and use

$$u^{y} \underset{y}{\circ} \emptyset \qquad \mu y.u\ .$$

6. FLOWCHARTS : MANNA'S FORMULAS

In the present section we consider charts with all operators unary. By a straightforward translation, we show how the operation computed by a chart can be expressed in second order logic, thus obtaining Manna's formulas for termination and correctness of flowcharts (see Manna [9], [10]).

A flowchart in usual notation contains both operations and predicates. We can eliminate the latter by introducing for each predicate p the partial identity functions I_p and $I_{\neg p}$ on the ranges of p and not p, respectively. Thus $I_p x$ is x if px holds, undefined otherwise; similarly for $I_{\neg p}$.

Now let (B,r,s) be a chart with all operators unary. We assume that there is one entry and one exit only and that both are determinate, i.e. that r and s are elements of B. Let A be the algebra in which we want to interpret the chart. According to the results of section 4, the operation f computed by the chart in A is

$$fx = y_m \ \underline{\underline{\text{where rec}}}\ y = dxy$$

(assuming that the chart has m nodes, associating y_m with the exit, and writing y for $(y_1,\ldots,y_m)$). The components $d_i xy$ of dxy are unions of terms that are either x or hy_j $(h\in\Omega)$.

Termination, partial correctness, correctness. We say that the chart terminates (under the given interpretation) for a given element $a\in A$, if $fa \neq \emptyset$; that it is partially correct for subsets $x, x'\subseteq A$, if $fx\subseteq x'$; that it is correct for $x, x'\subseteq A$, if $fa\cap x' \neq \emptyset$ for all $a\in x$.

Thus termination expresses that for each $a\in A$ there will be at least one result; partial correctness expresses that if there are results for an $a\in x$, then they are in x'. The definition of correctness is obvious for determinate interpretations, i.e. for interpretations for which fa is either $\emptyset$ or a single element. In the general case, it expresses only that some result is in x' for $a\in x$. (For a different definition in the nondeterminate case, which however can not be expressed in terms of f alone, see Manna [11].)

Translation into logic. We assume that the operations $h\in\Omega$ on A are (partial) determinate, i.e. $ha = \emptyset$ or $\in A$ for $a\in A$. For each $h\in\Omega$, we assume that we have a predicate p testing for the domain of h, i.e. $pa = (ha \neq \emptyset)$.

The system of sets $y = (y_1,\ldots,y_m)$ is the smallest solution of $y\supseteq dxy$; therefore $a\in A$ is in fx if it is in y_m for every y satisfying $y\supseteq dxy$. Thus we get the following translations:

$a \in fx$	$(\forall y)\ (y \supseteq dxy \rightarrow a \in y_m)$
$fa \neq \emptyset$	$(\forall y)\ (y \supseteq day \rightarrow y_m \neq \emptyset)$
$fx \subseteq x'$	$(\exists y)\ (y \supseteq dxy \wedge y_m \subseteq x')$
$(\forall a)\ (a \in x \rightarrow fa \cap x' \neq \emptyset)$	$(\forall y)\ (\forall a)\ (a \in x \wedge y \supseteq day \rightarrow y_m \cap x' \neq \emptyset)$

Now $y\supseteq dxy$ is a conjunction of conditions $y_i \supseteq hy_j$, $y_i \supseteq x$, for which we get the translations $(\forall a)\ (a \in y_j \cap pa \rightarrow ha \in y_i)$ and $(\forall a)\ (a \in x \rightarrow a \in y_i)$, respectively. Also $\emptyset$ and $\cap$ can easily be eliminated by first order formulas in ε. Finally, $a\in x$ becomes qa if we use predicates q of sets x. Therefore we have the result that termination, partial correctness, correctness can be expressed respectively as validity, satisfiability, validity of certain first order formulas.

Example 1. Consider the program (in Algol-like rather than in flowchart notation):

g;L:h;M:j; if p then (k; goto L); l; if q then (m; goto M); n .

The operation f computed by this chart is

$$fx = y_6 \text{ where rec } y_1 = x$$
$$y_2 = gy_1 \cup k(p \to' y_4)$$
$$y_3 = hy_2 \cup m(q \to' y_5)$$
$$y_4 = jy_3$$
$$y_5 = l(\neg p \to' y_4)$$
$$y_6 = n(\neg q \to' y_5)$$

where we have written $p \to' a$ for $I_p a$. By substitution we can simplify this to

$$fx = (\neg q \to' y_5) \text{ where rec } y_4 = j(h(gx \cup k(p \to' y_4)) \cup m(q \to' y_5))$$
$$y_5 = l(\neg p \to' y_4)$$

Note that we could eliminate also y_5 at the cost of duplicating some of the operators. Also, we could use * (iteration as familiar from regular expressions, see also end of section 1) and thus avoid any auxiliary definition, but at the cost of even more duplication. The formula whose validity expresses termination, i.e. $fa \neq \emptyset$, becomes

$$q_1 a \wedge$$
$$(\forall b)[(q_1 b \to q_2(gb)) \wedge (q_2 b \to q_3(hb)) \wedge (q_3 b \to q_4(jb)) \wedge$$
$$(q_4 b \wedge pb \to q_2(kb)) \wedge (q_4 b \wedge \neg pb \to q_5(lb)) \wedge$$
$$(q_5 b \wedge qb \to q_3(mb)) \wedge (q_5 b \wedge \neg qb \to q_6(nb))]$$
$$\to (\exists b)(q_6 b)$$

(Note that in the last formula we "collected arcs" with the same source, whereas for writing the equation system we had to collect arcs with the same target.)

Example 2. Consider the "count-down" program

L: if b=0 then (n:=pn; goto L)

where p is the predecessor function $\lambda n.n-1$. The operation computed by this program is

$$fx = y_2 \text{ where rec } y_1 = x \cup p(I_{\neq 0} \to' y_1)$$
$$y_2 = I_{=0}\, y_1$$

Termination becomes validity of

$$q_1 n \wedge (\forall m)[(q_1 m \wedge m \neq 0 \rightarrow q_1(pm)) \wedge (q_1 m \wedge m=0 \rightarrow q_2 m)] \rightarrow \exists m.q_2 m$$

i.e. equivalently of

$$q_1 n \wedge (\forall m)\ (q_1 m \wedge m \neq 0 \rightarrow q_1(pm)) \rightarrow q_1 0$$

i.e. the condition that every subset containing n and closed under p must contain 0. (To appreciate this result, recall that in fact we are asking which condition an arbitrary interpretation, with arbitrary constant 0 and function p, must satisfy in order that the above program terminates for argument n.)

CONCLUSION

We have obtained the regular-equals-recognisable theorem of generalised automata theory as a special case of two theorems on definable operations in general algebras. Definable operations arise from the familiar concept of derived operations by adding union and definition by recursion as rules for forming new operations. Our proofs consist in simple and intuitively obvious transformations on expressions allowing recursive definitions.

Definable operations are the operations computed by (generalised) flowcharts. Even for conventional flowcharts, the use of simultaneous recursion allows for a formulation of that operation which is simpler, and more directly tied to the structure of the flowchart, than the usual formulation obtained by the iteration operator*. By translating into predicate calculus, we obtained Manna's formulas for termination and correctness of flowcharts, thus showing a connection to what may have seemed an isolated technique.

It would be interesting to extend the study to the context-free case, i.e. to stack-automata on the automata side, to procedures on the programming language side. (De Bakker - Scott [1] give already results on procedures, also Manna's technique has been extended to deal with procedures by Manna-Pnueli [12].) Other relevant and interesting topics are parallel computation and Landin's generalised jumping operator J.

Relation to other work

The relevance of fixed point theorems to the theory of flowcharts has been realised by several authors at about the same time (see de Bakker - Scott [1], Park [13]). Scott also has the μ-operator and the elimination of simultaneous recursion.

Kleene's generalised theorem and its proof are due to Thatcher-Wright [15]. The motivation to replace their definition of regular set by something using derivable (and

going from there:definable) operations came at least partly from the use of "theories" in Eilenberg-Wright [5]; their paper also contains a theorem (theorem 2) that is essentially our lemma in section 4 on the minimal subalgebra of B×A (with A = T).

As mentioned earlier, our treatment of charts is due to ideas of Landin [8]. There are many other relationships to that paper. Perhaps the main difference in approach is that there is a homomorphism from formal objects (algebras, charts) to operations denoted by them is established, whereas our theorem II talks on operations only and confines the use of where - where rec - expressions to the informal level.

Acknowledgements. This is a report on research that I carried out while working for Peter Landin at Queen Mary College, London, under a Research Grant of the S.R.C. I would like to thank P. Landin for innumerably many suggestions and discussions. I would also like to acknowledge an early discussion I had with Malcom Bird on the subject.

REFERENCES

[1] J. W. de Bakker, D. Scott: A theory of programs. - IBM Seminar Vienna, August 1969 (unpublished).

[2] M. R. Bird: Binary relations and flow-diagrams. - Memorandum No.11, Sept. 1969, Computer and Logic Research Group, University of Swansea.

[3] R. M. Burstall, P. Landin: Programs and their proofs: an algebraic approach. - Machine Intelligence 4, (B. Meltzer, D. Michie Eds.), Edinburgh University Press 1969.

[4] P. M. Cohn: Universal algebra. - Harper Row. New York - London 1965.

[5] S. Eilenberg, J. B. Wright: Automata in general algebras. - Information and Control 11, 4, Oct. 1967.

[6] G. Grätzer: Universal algebra. - Van Nostrand, Princetown - London 1969.

[7] P. Landin: The mechanical evaluation of expressions. - Comp. Journal Jan. 1964.

[8] P. Landin: Minimal subalgebras and direct products - a scenario for the theory of computation. - Machine Intelligence 5.

[9] Z. Manna: Properties of programs and the first order predicate calculus. - JACM 16, 2, April 1969.

[10] Z. Manna: The correctness of programs. - J. Comp.Syst. Sciences 3, 1969.

[11] Z. Manna: The correctness of nondeterministic programs. - A. I. Memo No.95, Stanford University, 1969.

[12] Z. Manna, A. Pnueli: Formalisation of properties of recursively defined functions. - A. I. Memo No.82, Stanford 1969.

[13] D. Park: Some metatheorems for program equivalence proofs. - Machine Intelligence 5.

[14] A. Tarski: A lattice-theoretic fixpoint theorem and its applications. - Pacific J. Math. 5, p. 285-309, 1955.

[15] J. W. Thatcher, J. B. Wright: Generalised finite automata theory with an application to a decision problem of second-order logic. - Math. Syst. Theory, 2, 1, 1968.

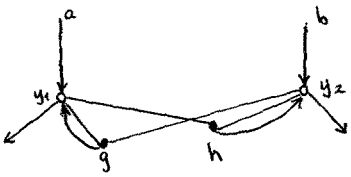

Fig.1.

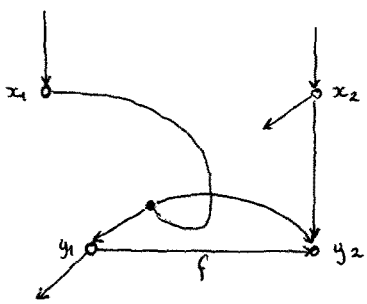

Fig.2.

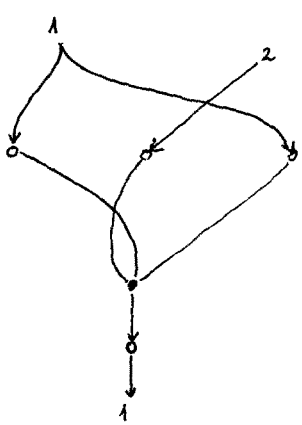

Fig.3.