EQUATIONS IN COMBINATORY ALGEBRAS

Erwin Engeler
ETH Zurich

1. The reduction of algorithmic problems to combinatorial equations

The algorithmic problems we have in mind are not decision problems such as: does x have property F ? Rather, we think of problems of the form: Find x such that $F(x)$. The latter is one of the archetypes of mathematical activities and, to obtain a realistic analysis, it is perhaps best to orient ourselves on classical examples of problems and solutions to visualize the spectrum of notions involved.

To fix ideas, consider the similarity type of relational structures $\underline{A} = \langle A,R,f,c \rangle$ with relation R , operation f , constant c , and let Γ be an axiom system specifying the actual class of models we have in mind.

<u>Algebraic problems</u> in Γ would be posed by equations of the form $t_1(x) = t_2(x)$, $t_1(x) = x$, $t_1(x) = c$, where t_1,t_2 are terms in the language of Γ . To solve the problem in $\underline{A}$ means to exhibit an element $a \in A$ which satisfies the equation in $\underline{A}$, to solve it <u>explicitly</u> means to describe a as a constant term in the language of Γ . Note that this term may be considered as a straight-line program using only assignments of the form $x_i := f(x_j,x_k)$. Some algebraic

problems are only solved in extensions of the original structure, e.g. $x^2 = 2$ is not solvable in $\underline{\mathbb{Q}}$ but only in $\mathbb{Q}(\sqrt{2})$. In this case, to solve the problem means to generate information about $\sqrt{2}$, e.g. $1 < \sqrt{2} < 2$, but also to be able to manipulate $\sqrt{2}$, at the least be able to perform the field operations involving $\sqrt{2}$. Thus, the solution $\sqrt{2}$ in a sense consists of a set X of formulas, and to manipulate $\sqrt{2}$ would mean to manipulate X . Before we make this any more definite, consider another type of problems.

<u>Basic problems</u> in Γ would be posed by basic, i.e. quantifier-free first-order, formulas $F(x)$ of the language of Γ . Classical examples are provided by elementary geometry. Take a geometry with two sorts, lines and points, and ask for the foci of a conic section given by five points. The theory of geometrical constructions with ruler and compass illustrate the notion of an <u>algorithmic solution:</u> it consists of exhibiting a program which constructs (using instructions corresponding to the construction tools) the required points. This can in the present case be done by a loop-free program, i.e. we are close to the case of explicit solution. But let us imagine a geometry of points and circles as sorts and ask for the angular bisector of two given lines. It can be shown, that all solution programs must contain a loop. Thus, if we identify "solution" with "solution program", the concept of algorithmic solution should incorporate a sufficiently large class of programs. As in the case of $\sqrt{2}$, there are classical examples of basic problems in geometry which have no solution in some given model (trisection of angles). Again, the solution would be a set X (of properties of the solution line), and would be algorithmic, if the solution program generated X and if it allowed to manipulate X in some manner associated to the relations and operations of the theory.

Algorithmic problems arise also in elementary geometry, the most celebrated one is the quadrature of the circle: We ask for a line segment, whose length is equal to the circumference of a given circle. This is not a basic problem (nor is it elementary, i.e. posed by a first-order formula of elementary geometry). But it can very easily be posed by a program (ruler and compass), which tests, whether the proposed line segment is indeed a solution: it does not terminate iff the line segment is a solution. Again, there is no explicit, but only an algorithmic solution to the problem.

The first purpose is now to create a framework, in which the common denominator of the above examples can be precisely formulated. The main new idea is that algorithms and algorithmic properties of structures should be included in the consideration of problems and solutions. We propose to do this by suitably enlarging the set of objects of the theory, concretely: to show that graph models of combinatory logic can serve as the basic structure in which the envisioned class of problems can be reformulated as equations.

Let A be any non-empty set. Consider the set $G(A)$ defined recursively by

$$\begin{aligned} G_0(A) &= A \\ G_{n+1}(A) &= G_n(A) \cup \left\{(\alpha \to b) : \alpha \subseteqq G_n(A),\ \alpha \text{ finite},\ b \in G_n(A)\right\} \\ G(A) &= \bigcup_n G_n(A) \end{aligned}$$

Graph algebras over A are constructed as sets of subsets of $G(A)$ closed under the following binary operation

$$M \cdot N = \left\{b : \exists \alpha \subseteqq N \cdot (\alpha \to b) \in M\right\} .$$

We now illustrate how graph algebras arise as formal counterparts to relational structures by associating to the field $\underline{\mathbb{Q}}$ its <u>state field</u> $\underline{\mathbb{Q}}^s$. For this purpose, let A be the set of all quantifier-free formulas of the first-order language of the field $\underline{\mathbb{Q}} = \langle \mathbb{Q},+,\cdot,-,^{-1},0,1,\leqq \rangle$.

(a) The state objects of $\underline{\mathbb{Q}}^s$ are exactly the subsets M of A for which there is an assignment of rational numbers to the variables occurring in M which satisfies all formulas in M.

(b) The state transformation objects of $\underline{\mathbb{Q}}^s$ are defined as follows $[x_i := x_j + x_k]$ consists of all $(\alpha \to b)$, where $\alpha \subseteqq A$, $b \in A$ with the following properties: α is consistent with the theory Γ of $\underline{\mathbb{Q}}$; if $i \neq j,k$ and x,y,z do not occur in α or b and b' results from substituting x for x_i, y for x_j and z for x_k then b' is a consequence of Γ, α and the equations $x = x_j + x_k$, $y = x_j$, $z = x_k$; if $i = j$, say, then the equation $y = x_j$ is to be suppressed, similarly for $i = k$ and $i = j = k$.
The state transformers corresponding to the other instructions, $[x_i := x_j \cdot x_k]$, $[x_i := -x_j]$, $[x_i := x_j^{-1}]$ are defined in like manner.
The truth value objects are defined using the important auxilliary constructs

$$K = \left\{ \{\alpha\} \to (\phi \to a) \ : \ a \in G(A) \right\} ,$$

$$I = \left\{ \{\alpha\} \to a \ : \ a \in G(A) \right\} ,$$

$$\text{true} = K ,$$

$$\text{false} = K \cdot I .$$

Then $[x_i \leqq x_j]$ consists first of all $(\alpha \to b)$ where $\alpha \subseteqq A$, $x_i \leqq x_j$ is a consequence of Γ and α and

$b \in K$ = true and second of all $(\alpha \to c)$ where $x_j > x_i$ is a consequence of α and Γ and $c \in K \cdot I$ = false.

(c) Finally $\underline{\mathbb{Q}}^s$ is the graph algebra generated by the objects introduced in (a) and (b) above.

We observe that all the objects of $\underline{\mathbb{Q}}^s$ are recursively enumerable subsets of $G(A)$. Indeed the generators are, and if M and N are recursively enumerable then so is $M \cdot N$. Thus $\underline{\mathbb{Q}}^s$ forms what we would like to call a computable graph algebra.

A first extension of $\underline{\mathbb{Q}}^s$ is suggested by algebraic problems. It consists in the adjunction of the composition object B whose defining property is $BMNL = M(NL)$ for all $M,N,L \subseteqq G(A)$. Such an object can easily be constructed for graph algebras, e.g.

$$B = \left\{(\alpha \to (\beta \to (\gamma \to a))) \ : \ a \in \alpha(\beta\gamma), \alpha, \beta, \gamma \subseteqq G(A), \text{ finite}\right\} .$$

Indeed, as shown in [4], every object F given by a defining relation $F \cdot x_1 \cdot \ldots \cdot x_n = t(x_1, \ldots, x_n)$ where t is any combination of the x_i by means of $\cdot$ can be realized by a (recursively enumerable) set of elements of $G(A)$. Observe now, that any algebraic problem $p(x) = 0$ over $\underline{\mathbb{Q}}$ can be reformulated as an equation

$$P \cdot X = \text{true} ,$$

where P is composed of state transformers and B. In general, solutions X are not among the state objects of $\underline{\mathbb{Q}}^s$ since as state objects they should include the equation $x^2 = 0$ which is impossible in $\underline{\mathbb{Q}}$. However, algorithms such as Newton's method generate a set of nested intervals.

This generating process can be reformulated as one that generates the set $\left\{ q_i < x < p_i \ : \ q_i, p_i \text{ the endpoints of the Newton intervals} \right\}$ which is a solution. Now, the algorithm underlying Newton's method employs program connectives other than composition, essentially a <u>while</u> loop. To mirror this construction algorithm by an object in a graph algebra we need to have an object $[\underline{\text{while}}] \subseteqq G(A)$ with the appropriate properties. Newton's method then is realized in the graph algebra by a composite object [newton] with the property $P \cdot \left([\text{newton}] \cdot \{x^2 = 2\} \right) = \text{true}$ and $[\text{newton}] \cdot \{x^2 = 2\}$ contains a set of convergent nested intervals (as formulas $p_i < x < q_i$).

Observe, that we have created a sequence of adjunctions to $\underline{\mathbb{Q}}^s$, viz. $\underline{\mathbb{Q}}^s \subseteqq \underline{\mathbb{Q}}^s(B) \subseteqq \underline{\mathbb{Q}}^s\left(B, \{x^2 = 2\}, [\underline{\text{while}}]\right)$ which not only let us encompass new solutions but also new problems, in fact algorithmic problems of the kind envisioned above.

There are a number of obvious questions that arise here. First, clearly, the state field approach to computing is one that has the promise of actual realization on the computer. This has in part been accomplished by Fehlmann [5], [6] who has written a system CONCAT which takes a large sublanguage of PASCAL, translates its programs into graph-algebra objects and performs the PASCAL computations with such objects. The effect of this method is to obtain (from PASCAL algorithms that are correct only for infinitely exact reals) a corresponding graph-algebra computation which is correct to any desired accuracy.

Second, our approach gives a new point of view in the logic of programs. Indeed, it is of considerable interest to investigate the definitional power of program connectives.

This translates here into the algebraic question of comparing extensions of $\underline{\mathfrak{Q}}^{s}$ by B, [while] and other objects corresponding to program connectives; e.g. whether [recursion] would give a proper extension of $\underline{\mathfrak{Q}}^{s}$(B , [while]) , and whether such answers depend, and how, on the original state structure, (here $\underline{\mathfrak{Q}}^{s}$). Some such effects are well known.

Third, there now arise quite general questions of the following nature: Let $\underline{B}$ be any graph algebra and let $t_1(x) = t_2(x)$ be an equation. Can this equation be solved (in some extension of) $\underline{B}$ and if so, by what means can the graph-algebra object x be obtained? The remainder of this paper addresses itself to some aspects of this question in a more abstract setting.

2. Algebraic extensions of graph algebras

It is well-known (see e.g. [4]) that the graph algebra consisting of all subsets of $G(A)$ constitutes a combinatory algebra, i.e. there are subsets $S,K \subseteq G(A)$ such that $Kxy = x$, $Sxyz = xy(xz)$ for all $x,y,z \subseteq G(A)$. Indeed, one can also find $L \subseteq G(A)$ such that $(L\cdot x)\cdot y = x\cdot y$ and $\forall z(x\cdot z = y\cdot z). \supset L\cdot x = L\cdot y$, which makes it a combinatory model, (even a stable one, by managing $L\cdot L = L$). As we have seen in the first section of this paper, some mathematically and computationally interesting structures can be realized as subalgebras of such combinatory models. Indeed, as has also been shown in [4], every algebraic structure can be isomorphically embedded in an appropriate combinatory model.

Taking into account this embeddability, we now consider the question of solving algebraic equations in the richer context of combinatory algebras, specifically the above so-called graph models. Let therefore $X_1,\ldots,X_m$ be variables, $A_1,\ldots,A_k$ be constants denoting elements of $\underline{G(A)}$, i.e. subsets of $G(A)$. The problem to be solved is presented by one or more equations in the "unknowns" X_i and "parameters" A_i, written in the form $t_j(X_1,\ldots,X_m) = t'_j(X_1,\ldots,X_m)$, where the t_j, t'_j are composed by the binary operation of our structure $\underline{G(A)}$. A is assumed countable.

For convenience, and without loss of generality, we consider instead of $G(A)$ the following set L of "lists". The set L is obtained as a set of syntactical terms constructed by means of a binary syntactical operation c and unary operations $f_1,\ldots,f_n$ as follows: The empty list O is a list. If u and v are lists, then $c(u,v)$ and $f_1(u),\ldots,f_n(u)$ are lists.

If M is any set of lists and u is a list, we define $u \leqq M$ recursively by:

(a) $O \leqq M$.

(b) $c(u,v) \leqq M$ iff $u \in M$ and $v \leqq M$.

With this notation we introduce the application of the combinatory algebra $\underline{L}$ (which consists of subsets of L) by:

$$M \cdot N = \left\{ u : \exists v.\ c(u,v) \in M \wedge v \leqq N \right\} .$$

We are now ready to state the main result of this paper. Let $A_1,\dots,A_k$ be recursively enumerable subsets of L, let E be a set of equations with parameters A_i and unknowns $X_1,\dots,X_m$, where length$(E) \leqq n$, the number of unary syntactical operations f_i. A solution of E consists of sets $A_1',\dots,A_k'$, $X_1',\dots,X_m'$ with $A_i' \supseteqq A_i$ such that E is satisfied.

THEOREM. If E is solvable, then E has a recursively enumerable solution.

The proof of the above theorem is an application of logic programming, (see e.g. [2],[3],[7]). This framework allows us to state the recursive enumeration of the parameter sets A_i and the solution conditions of E conveniently as a set of (universal) first-order formulas. Our use of logic programming may be formulated as the following lemma.

LEMMA. Let $W \subseteqq L$ be recursively enumerable. Then there exist predicate symbols $P_1,\dots,P_m$, W, and a_i quantifier-free conjunction of positive Horn formulas, ϕ_w, such that for all $w \in L$: $\forall \vec{x}\ \phi_w(x) \wedge \neg W(w)$ is inconsistent iff $w \in W$. $\square$

(Recall that for atomic formulas α_i, β we call $\neg\alpha$ a negative, and $\alpha_1 \wedge \dots \wedge \alpha \supset \beta$ and α_i positive Horn formulas). By Herbrand's theorem, we have:

COROLLARY. $w \in W$ iff there exists n and $\vec{u}_1,\dots,\vec{u}_n$ in L such that $\bigwedge_{i=1}^{n} \phi_W(\vec{u}_i) \wedge \neg W(w)$ is inconsistent, i.e. iff $W(w)$ is provable from $\phi_w(\vec{u}_1),\dots,\phi_w(\vec{u}_n)$ in propositional logic.

Logic programming consists in essence of a systematic way of generating terms (here "lists") and substituting them in sets of copies of the set of clauses ϕ_W that constitutes the "logic program" for W. This is usually done by some refined methods of unification and resolution on which rest the practicability of the approach. We refer the reader to the literature cited above. We now sketch the proof of the theorem using the equation

$$A\cdot X = B\cdot(X\cdot C)$$

Let A,B,C be recursively enumerable sets of lists and let the corresponding formulas ("logic programs") be given as ϕ_A, ϕ_B and ϕ_C. To the conjunction of these formulas we add the following formulas, corresponding roughly to $D = A\cdot X$, $E = X\cdot C$, $F = B\cdot E$:

$$\begin{cases} A(c(x,y)) \wedge \overline{X}(y). \supset D(x), & D(x) \supset. \overline{X}(f_1(x)) \wedge A(c(x,f_1(x))), \\ \overline{X}(c(x,y)). \equiv X(x) \wedge \overline{X}(y), & \overline{X}(0) \end{cases}$$

$$\begin{cases} B(c(x,y)) \wedge \overline{E}(y). \supset F(x), & F(x). \supset \overline{E}(f_2(x)) \wedge B(c(x,f_2(x))), \\ \overline{E}(c(x,y)). \equiv E(x) \wedge \overline{E}(y), & \overline{E}(0) \end{cases}$$

$$\begin{cases} X(c(x,y)) \wedge \overline{C}(y). \supset E(x), & E(x) \supset. \overline{C}(f_3(x)) \wedge X(c(x,f_3(x))), \\ \overline{C}(c(x,y)). \equiv C(x) \wedge \overline{C}(y), & \overline{C}(0) \end{cases}$$

The equation itself induces us to add conjunctively

$$\Big\{\ D(x) \equiv F(x) \quad .$$

Let $\psi(x,y)$ be the resulting formula.

The initial step of the algorithm that produces $X \subseteq L$ consists of refuting

$$\forall z\ \forall xy\Big(\psi(x,y) \wedge \neg X(z)\Big)$$

by a counterexample $X(w_1)$ (which is possible because we assume of course that the equation has a nontrivial solution).

In the k-th iteration step we first make sure that all of A,B,C are eventually taken in consideration. This is done by adding to ψ formulas $A(u_i')$, $B(u_i'')$, $C(u_i''')$ for the first k elements of A,B,C resepectively. We also add $X(w_1),\ldots,X(w_{k-1})$ for the w_i obtained at earlier steps. The resulting formula

$$\forall z\ \forall xy\Big(\psi(x,y) \wedge z \neq w_1 \wedge \ldots \wedge z \neq w_{k-1} \wedge \neg X(z)\Big)$$

is again provided with a counterexample (if one exists) otherwise, i.e. if the search for a counterexample does not succeed, we have already finished constructing our solution set X , because the set X of lists w_i for which the formula $X(w_i)$ is ever added to ψ solves the equation. Namely: let $A',B',\ldots$ be the sets of $v \in L$ for which $A(v)$, $B(v)$, ... can be proved from ψ at some stage of the algorithm. Then, by construction, $A' \cdot X = D'$, etc., hence $A' \cdot X = B' \cdot (X' \cdot C')$ as claimed.

Remarks

1. For some parameter values A_i it is possible to restrict A_i' to equal A_i , e.g. for the combinators K , L and S . At the time of this writing we are not yet able to determine a general criterion for this behaviour.

2. In $\underline{L}$ it is possible to reduce finite sets of equations to a normal form as follows:

LEMMA. Let E be a finite set of equations of the form

$$y_1 \ldots y_n.\ t_i(A_1,\ldots,A_k,X_1,\ldots,X_m,y_1,\ldots,y_n) = t_i'(A_1,\ldots,y_n)$$

with parameters $A_i \subseteq L$, recursively enumerable, unknowns X_i and variables y_i. Then there are recursively enumerable sets $A,B \subseteq L$ such that solving E is equivalent to solving the single equation $A \cdot Z = B \cdot Z$ for Z. □

3. The present approach to models of combinatory logic owes much to the pioneering work of Plotkin [10], Scott (e.g. [11]), Meyer [9], Longo [8] and Barendregt (e.g. [1]) and to conversations with these authors.

References

[1] Barendregt, H., Lambda Calculus and its Models. To appear in the proceedings of the Logic Colloquium'82, Florence, Italy.

[2] Clark, K.L. and Tärnlund, S.-A. (editors), Logic Programming. Academic Press, 1982.

[3] Clocksin, W.F. and Mellish, C.S., Programming in Prolog. Springer-Verlag, 1981.

[4] Engeler, E., Algebras and Combinators. Algebra Universalis 13 (1981), 389-392.

[5] Fehlmann, T., Theorie und Anwendung des Graphmodells der kombinatorischen Logik. Berichte des Instituts für Informatik, ETH Zürich, Nr. 41, 1981.

[6] Fehlmann, T., Concat Reference Manual and Implementation Notes. ETH Zürich, 1981, 48 pp.

[7] Kowalski, R.A., Logic for Problem Solving. North-Holland Publ. Co., 1979.

[8] Longo, G., Set-theoretical Models for the λ-calculus: Theories, Expansions, Isomorphisms. To appear in the Annals of Mathematical Logic.

[9] Meyer, A.R., What is a Model of the Lambda Calculus? Preprint MIT/LCS/TM-201, 1981.

[10] Plotkin, G.D., A Set-theoretical Definition of Application. Memorandum MIP-R-95, University of Edinburgh, 1972.

[11] Scott, D.S., Relating Theories of the λ-calculus. In: Essays on Combinatory Logic, Lambda Calculus and Formalism (to H.B. Curry), Seldin & Hindley, eds. Academic Press, New York, 1980, pp. 403-450.