MORE ON ADVICE ON STRUCTURING COMPILERS AND PROVING THEM CORRECT

James W. Thatcher, Eric G. Wagner and Jesse B. Wright

Mathematical Sciences Department
IBM Thomas J. Watson Research Center
Yorktown Heights, New York 10566
USA

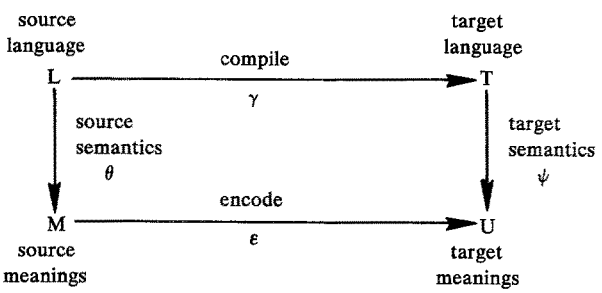
1. INTRODUCTION

The purpose of this paper is to affirm and applaud the advice given by F. L. Morris (1973) at the Second SIGACT/SIGPLAN Symposium on Principles of Programming Languages and to correct, refine, and complete the example he gave there.

The goal, first announced by McCarthy, is to make compilers for high level programming languages completely trustworthy by proving their correctness. Morris (1973) stated his belief (shared by many) that the compiler correctness problem is much less general and better structured than the unrestricted program correctness problem.

The essence of Morris' advice was that a proof of compiler correctness should be a proof that a diagram of the form[†]

(★)



commutes; that the corners of the diagram are not just sets but are many-sorted (heterogeneous) algebras and that the arrows are homomorphisms.

This paper can be seen as the fourth in the sequence: McCarthy and Painter (1967), Burstall and Landin (1970) and Morris (1973). At each step the content of (★) has become more algebraic and the example source language richer. Ours is not the last step! Much can be done to improve the picture, including a thorough analysis of the primitives used in the semantics of both source and target languages along the lines of Mosses (1978, 1979). The correctness proof should be mechanical; but the algebraic preliminaries must be further developed.

[†] Morris' diagram had $\delta: U \to M$ along the bottom, though in the text he uses $\varepsilon: M \to U$.

Morris observed that the source language, being described by a context free grammar, determined an initial many-sorted (heterogeneous) algebra. This correspondence is discussed in detail in ADJ (1975); if G is the grammar and N is its set of non-terminals, then G is viewed as an N-sorted operator domain where the productions are the operator symbols. T_G is the initial G-algebra and its carrier of sort $A \in N$ is the set of all parse trees from non-terminal A.

Recall that T_G being initial means that there is a unique homomorphism from it to any other algebra with operator domain G. This is how the top and left side of the diagram (★) are determined; L is T_G and M and T are G-algebras -- then γ and θ are unique homomorphisms. Initiality is also the method of correctness proof, for if ψ and ε are also homomorphisms, then $\gamma \circ \psi = \theta \circ \varepsilon$ by uniqueness. This is an extremely powerful methodology; no "structural induction" is required for the definition of the arrows or the proof.

So to describe the source semantics (the left side of the diagram) we need only define a G-algebra M, that is, carriers corresponding to the non-terminals and operations corresponding to the productions. Morris, "as a concession to readability," combined the specification of θ and M's operations in a "conventional style of recursive function definition, following the notation of Scott and Strachey (1970)." However we claim the result is *not* more readable for two reasons; first, combining with the definition of θ is just more notation -- θ is uniquely determined, and, second, the algebraic operations (composition, tupling, product, etc.) have not been separated out from the "local" operations, those involved with manipulating environments or "adding numbers." For example, for assignment, our semantic line is (see (M2)):

$$(\alpha)x{:=}_M = \alpha \circ assign_x$$

where $\alpha: E \to E \times V$ (E = environments and V = values) and $assign_x$ is the obvious function from $E \times V$ to E. (environments). Were we not being pedantic about writing arguments to the left of functions, that line would look even more familiar and simpler:

$$x{:=}_M(\alpha) = \alpha \circ assign_x.$$

In contrast Morris writes:

$$\theta[x{:=}r] = (\lambda a \lambda e.\lambda w.w{=}\theta[x] \to a, e(w)) * \theta[r]$$

where $p*q = \lambda x.p(q(x)_1)(q(x)_2)$! This is an incredible difference. It comes from our attempt to isolate the fundamental operations used in the semantic definitions just as Mosses (1980) wants to do with his semantic data types.

Our treatment differs substantially from that of Morris in that we have succeeded in making the right-hand side of (★) algebraic. This is what Morris wanted to do, but his algebraic model of flow charts was too unwieldy. In particular, we do not see the justification for his claim that a semantic homomorphism is determined by specifying the effect of the homomorphism on the individual instructions. Recognizing fundamental operations for building up flow charts (Section 5) and uniqueness of interpretation (Section 7) are crucially important contributions of Elgot (1973).

Thus, we take for the target language, an algebra T_0 of flow charts (actually a category) whose operations are things like parallel and serial composition and iteration, and whose individual instructions manipulate a stack and a "memory." The semantics of this category of flow charts is uniquely determined by

the interpretations of the flow chart primitives. The semantic target (U_0) is a category of meanings for those flow charts (actually an algebraic theory in the sense of Lawvere (1963)).

Then we extract from T_0 a G-algebra T, defining the operations of T in terms of the operations of T_0. (As the TCS referee emphasized, the "compiler writer must be warned that the extraction of a G-algebra from T_0 (the true target language) is, in fact, the very difficult work he usually calls 'compiler design' or 'implementation choices.'" It is the essence of compiler construction.) By initiality this gives the compile function γ:L→T and it also *immediately* determines an algebra U, extracted from U_0, and a homomorphism ψ from T to U. These arrows are each uniquely determined by the interpretations of certain primitives. All that is left is the "bottom line," ε:M→U. Given the (simple) definition of ε from the carrier of M to the carrier of U we have to prove that ε is a homomorphism, that is, that it preserves all the operations of M. Once this is done, the compiler correctness proof is complete for, by initiality, $\gamma \circ \psi = \theta \circ \varepsilon$.

As Barry Rosen has pointed out to us, commuting of (★) is not, in itself, "compiler correctness." T and U could be one-point algebras and γ, ψ, and ε, the unique homomorphisms to those one-point algebras resulting in a commuting square. At first look this is somewhat misleading because we should assume that the source language (with semantics), θ:L→M, and the target language are given. But, as indicated above, the algebraic structure of the target language is different from that of the source. The process of constructing the compiler consists of extracting an algebra T from the target language T_0; we could foolishly extract a one-point algebra, then U would automatically be a one-point algebra and Rosen's point is reinstated.

One possible way around the degenerate case of one-point algebras, suggested by Rosen, is to require the encoding (ε) to be injective (it is in our case). Then, as Steve Bloom has argued [personal communication], commutativity of ★ would say that there is a subalgebra of U which is isomorphic to M and up to this isomorphism a source program and its compiled target program have the same meaning. This is certainly a sufficient condition for compiler correctness. Several (including Steve Bloom and Barry Rosen) have argued, that it is necessary; you do *not* want to identify semantic objects in a translation of the source language. Two programs with distinct semantics must have, at this level of abstraction, distinct target meanings. But is it conceivable that distinct program phrases might have the same target semantics. We are just not sure at this time that injectivity is necessary.

Both Gaudel (1980, 1980a) and Mosses (1980) want to present source and (in effect) target semantics in terms of abstract data types. Then correctness becomes a property of the implementation or representation of the source data type in terms of the target data type. Mosses clearly requires this implementation to be injective; it is the part of his proof that is as long as ours.

Although our example language is similar to that of Milner and Weyrauch (1972)[†] and Milner (1976), our approach is different because we are explicitly avoiding the lambda calculus and because their target semantics is interpretive. In another treatment, and a concise one also, Germano and Maggiolo-Schettini (1975) present a compiler from a simple source language which computes sequence-to-sequence partial

---

[†] Milner [personal communication] commented on the Milner and Weyrauch (1972) proof: "... we could only think clearly enough to do our proof at all on the machine by structuring it algebraically." (See page 58 of their paper.)

recursive functions, to a target language which is a modification of Markov normal algorithms. Contrary to the approach advocated by Morris and by us, semantics given by Germano and Maggiolo-Schettini is not homomorphic: "semantics consists in a correspondence between syntactic objects (strings of symbols) and mathematical objects ..." But both source and target semantics *could be* homomorphic and it would be interesting to see how this reformulation would change their correctness proof.

Our treatment differs from Morris (1973) in one other less significant respect. We make the advice that the starting point for the semantic definition should be an abstract data type in the sense of ADJ (1976,1976a).

This paper presumes familiarity with many-sorted algebras, categories, and algebraic theories, but we hope that it can be read without detailed knowledge of those concepts. It is our intention that the example will prove potent enough to convince the reader of the importance of the algebraic ideas; that they are worth the investment of time and energy to obtain even better understanding.

2. THE UNDERLYING DATA TYPE

Let Σ be the following $\{int,Bool\}$-sorted signature for integer and Boolean valued expressions.

$\Sigma_{int,\lambda}=\{0,1\}$ $\quad \Sigma_{int,int}=\{^{-},Pr,Su\}$ $\quad \Sigma_{int,int\ int}=\{+,-,\times\}$

$\Sigma_{Bool,\lambda}=\{tt,ff\}$ $\quad \Sigma_{Bool,Bool}=\{\neg\}$ $\quad \Sigma_{Bool,Bool\ Bool}=\{\wedge,\vee\}$

$\Sigma_{Bool,int}=\{even\}$ $\quad \Sigma_{Bool,int\ int}=\{\leq,\geq,EQ\}$ $\quad \Sigma_{int,Bool\ int\ int}=\{cond\}.$

All other $\Sigma_{s,w}$ are empty. $T_{\Sigma,int}$ is the set (or algebra) of integer valued expressions and $T_{\Sigma,Bool}$ is the set of Boolean valued expressions. The underlying data type (an $\{int, Bool\}$-sorted algebra S) for our simple programming language is the abstract data type[†] determined by the signature Σ together with axioms E consisting of at least (the correctness of these axioms is not at issue for this paper) axioms E1-E27 below. Assuming those axioms *are* correct (in the strong sense of ADJ (1976a)), we can take $S_{int} = \mathbb{Z}$ (the integers) and (for technical reasons) $S_{Bool} = [2] = \{1,2\}$ (with $tt_S = 2$).

(E1) $Pr(Su(x)) = x$ $\quad$ (E2) $Su(Pr(x)) = x$ $\quad$ (E3) $Su(0) = 1$

(E4) $\neg(tt) = ff$ $\quad$ (E5) $\neg(ff) = tt$

(E6) $b \wedge tt = b$ $\quad$ (E7) $b \wedge ff = ff$ $\quad$ (E8) $b \vee b' = \neg(\neg b \wedge \neg b')$

(E9) $x+0 = x$ $\quad$ (E10) $x+Su(y) = Su(x+y)$

(E11) $x-0 = x$ $\quad$ (E12) $x-Su(y) = Pr(x-y)$

(E13) $x \times 0 = 0$ $\quad$ (E14) $x \times Su(y) = (x \times y)+x$ $\quad$ (E15) $^{-}(x) = 0-x$

(E16) $cond(tt,x,y) = x$ $\quad$ (E17) $cond(ff,x,y) = y$

(E18) $x \leq x = tt$ $\quad$ (E19) $1 \leq 0 = ff$ $\quad$ (E20) $x \leq y = Su(x) \leq Su(y)$

(E21) $x \leq y = tt \Rightarrow x \leq Su(y) = tt$ $\quad$ (E22) $x \leq y = ff \Rightarrow S(x) \leq y = ff$

(E23) $EQ(x,y) = (x \leq y) \wedge (y \leq x)$ $\quad$ (E24) $x \geq y = EQ(x,y) \vee \neg(x \leq y)$

(E25) $even(0) = tt$ $\quad$ (E26) $even(1) = ff$ $\quad$ (E27) $even(x) = \neg even(Su(x))$

[†] See, for instance, Zilles (1974), Guttag (1975) or ADJ (1976a).

3. THE LANGUAGE L

Our programming language is essentially the one employed by Morris (1973). As such, it is a slight enrichment of the language used as an example by Milner (1976). Our grammar will have non-terminals {<st>,<ae>,<be>} for "statements", "arithmetic expressions" and "Boolean expressions." The terminals include the symbols in the signature Σ above, plus those other letters in boldface occurring in the productions below. Further, we assume given a set X of variables or identifiers.

We list the productions of G giving each a name which we can use in defining the semantic algebra. Thus, for example, when G is viewed as an operator domain, ifthenelse is an operator symbol to denote a function that takes three arguments of sorts <be>,<st>,<st>, respectively, and yields a result of sort <st>. Similarly result takes two arguments of sort <st> and <ae> and yields a result of sort <ae>.

(L1)	continue	<st> ::= **continue**	
(L2)	x:=	<st> ::= **x:=**<ae>	For $x \in X$
(L3)	ifthenelse	<st> ::= **if**<be>**then**<st>**else**<st>	
(L4)	;	<st> ::= <st>;<st>	
(L5)	whiledo	<st> ::= **while**<be>**do**<st>	
(L6)	c	<ae> ::= **c**	For $c \in \Sigma_{int,\lambda}$
(L7)	x	<ae> ::= **x**	For $x \in X$
(L8)	aop1	<ae> ::= **aop1**<ae>	For $aop1 \in \Sigma_{int,int}$
(L9)	aop2	<ae> ::= <ae>**aop2**<ae>	For $aop2 \in \Sigma_{int,int\ int}$
(L10)	cond	<ae> ::= **if**<be>**then**<ae>**else**<ae>	
(L11)	result	<ae> ::= <st>**result**<ae>	
(L12)	letx	<ae> ::= **let x be**<ae>**in**<ae>	For $x \in X$
(L13)	bc	<be> ::= **bc**	For $bc \in \Sigma_{Bool,\lambda}$
(L14)	prop	<be> ::= **prop**<ae>	For $prop \in \Sigma_{Bool,int}$
(L15)	rel	<be> ::= <ae>**rel**<ae>	For $rel \in \Sigma_{Bool,int\ int}$
(L16)	bop1	<be> ::= **bop1**<be>	For $bop1 \in \Sigma_{Bool,Bool}$
(L17)	bop2	<be> ::= <be>**bop2**<be>	For $bop2 \in \Sigma_{Bool,Bool\ bool}$

4. SOURCE LANGUAGE SEMANTICS, THE ALGEBRA M.

Now we want to define the semantic algebra M. For this we need the set Env of "environments," Env = $[X \to \mathbb{Z}]$. Then the three carriers are:

$$M_{<st>} = [Env \multimap Env] \quad M_{<ae>} = [Env \multimap Env \times \mathbb{Z}] \quad M_{<be>} = [Env \multimap Env \times [2]].$$

Here $[A \to B]$ is the set of (total) functions from A to B and $[A \multimap B]$ is the (po)set of partial functions from A to B.

The definitions of the seventeen operations on M (corresponding to the grammar's seventeen productions) involve certain primitive operations on M's carriers, including assign and fetch, along with standard (and some not so standard) operations on functions such as composition, tupling and iteration. Thus before

presenting M's operations we must familiarize the reader with what is, in effect, our metalanguage for giving the definition of M. The reader should be advised that this "metalanguage" will be used throughout this paper.

We first list the primitive operations:

$$\text{assign}_x\text{:Env}\times\mathbb{Z}\to\text{Env} \qquad (y)\langle e,v\rangle\text{assign}_x = \begin{cases} v \text{ if } y=x \\ (y)e \text{ if } y\neq x \end{cases}$$

$$\text{fetch}_x\text{:Env}\to\text{Env}\times\mathbb{Z} \qquad (e)\text{fetch}_x = \langle e,(x)e\rangle$$

We also have available all the operations σ_S, for $\sigma\in\Sigma$, from Section 2; e.g., $+_S$ is addition on the integers.

Now for the more general considerations. The set [2] was used in Section 2; [n] is the set $\{1,2,\ldots,n\}$. For both total and partial functions, we will write $f:A\to B$ to designate source and target, function arguments will *usually* be written on the left as in (a)f, and we will explicitly write $\circ$ for the operation of function composition whose arguments are written in diagrammatic order: if $f:A\to B$ and $g:B\to C$ then $f\circ g:A\to C$. 1_A is the identity function on the set A (for $f:A\to B$, $1_A\circ f = f = f\circ 1_B$).

Given two (partial) functions, $f_i:A\to B$, define the *source tuple*, $(f_1,f_2):A\times[2]\to B$, by

$$\langle a,i\rangle(f_1,f_2) = (a)f_i.$$

Define the *sum*, $f_1+f_2:A\times[2]\to B\times[2]$, of functions $f_i:A\to B$ for $i\in[2]$ by:

$$\langle a,i\rangle(f_1+f_2) = \langle(a)f_i,i\rangle.$$

If $\iota_i:B\to B\times[2]$ is the injection sending $b\in B$ to $\langle b,i\rangle$, for $i\in[2]$, then $f_1+f_2=(f_1\circ\iota_1,f_2\circ\iota_2)$. $B\times[2]$ is the disjoint union, sum or coproduct of B with itself, and more generally $B\times[n]$ is the coproduct of B with itself n times (n disjoint "copies" of B); $\iota_i:B\to B\times[n]$ sends b to $\langle b,i\rangle$, for $i\in[n]$. Context will usually distinguish the source of an injection and for this paper, the target will always be clear. When necessary to distinguish sources, we will write $\iota_j^B:B\to B\times[n]$.

Given a partial function $f:A\to A\times[2]$, define the *iterate*, $f^\dagger:A\to A$, to be the least upper bound (i.e. union) of the sequence $f^{(k)}$ defined by:

$$f^{(0)} = \emptyset$$
$$f^{(k+1)} = f\circ(f^{(k)},1_A),$$

where $\emptyset$ is the empty partial function from A to A. Iteration is the least familiar operation that we use; it replaces the fixed-point operator (Y) of other semantic definitions. Say $f:\text{Env}\to\text{Env}\times[2]$ is a function that takes an environment e, creates a new environment e′ and then performs some test, giving $\langle e',1\rangle$ if the test is false and $\langle e',2\rangle$ if the test is true. Then the function $f^\dagger:\text{Env}\to\text{Env}$ is the function corresponding to our intuition, "do f until its test is true."

Given (partial) functions $f_i:A\to B_i$, define the *target tuple*, $[f_1,f_2]:A\to B_1\times B_2$, by:

$$(a)[f_1,f_2] = \langle(a)f_1,(a)f_2\rangle.$$

Note that if either f_1 or f_2 is undefined at a, then $[f_1,f_2]$ is undefined at a. The projection function $\pi_i:A_1\times\ldots\times A_n\to A_i$ takes $<a_1,\ldots,a_n>$ to a_i. Given functions $f_i:A_i\to B_i$, define their *product*, $f_1\times f_2:A_1\times A_2\to B_1\times B_2$, by:

$$<a_1,a_2>(f_1\times f_2) = <(a_1)f_1,(a_2)f_2>.$$

Paralleling the sum case above, the product of functions is defined in terms of target tupling and projections: $f_1\times f_2=[\pi_1\circ f_1,\pi_2\circ f_2]$.

Now for the definitions of M's operations; τ,τ_1,τ_2, range over $M_{<st>}$; α,α_1,α_2 range over $M_{<ae>}$; and, β,β_1,β_2 range over $M_{<be>}$.

(M1) $\quad \mathsf{continue}_M = 1_{Env}$

(M2) $\quad (\alpha)\mathsf{x:=}_M = \alpha\circ\mathsf{assign}_x$

(M3) $\quad (\beta,\tau_1,\tau_2)\mathsf{ifthenelse}_M = \beta\circ(\tau_1,\tau_2)$

(M4) $\quad (\tau_1,\tau_2)\mathsf{;}_M = \tau_1\circ\tau_2$

(M5) $\quad (\beta,\tau)\mathsf{whiledo}_M = (\beta\circ(\tau+1_{Env}))^\dagger$

(M6) $\quad \mathsf{c}_M = 1_{Env}\times c_S$

(M7) $\quad \mathsf{x}_M = \mathsf{fetch}_x$

(M8) $\quad (\alpha)\mathsf{aop1}_M = \alpha\circ(1_{Env}\times \mathsf{aop1}_S)$

(M9) $\quad (\alpha_1,\alpha_2)\mathsf{aop2}_M = \alpha_1\circ(\alpha_2\times 1_{\mathbb{Z}})\circ[\pi_1,\pi_3,\pi_2]\circ(1_{Env}\times \mathsf{aop2}_S)$

(M10) $\quad (\beta,\alpha_1,\alpha_2)\mathsf{cond}_M = \beta\circ(\alpha_1,\alpha_2)$

(M11) $\quad (\tau,\alpha)\mathsf{result}_M = \tau\circ\alpha$

(M12) $\quad (\alpha_1,\alpha_2)\mathsf{letx}_M = \mathsf{fetch}_x\circ[(\alpha_1\circ\mathsf{assign}_x\circ\alpha_2)\times 1_{\mathbb{Z}}]\circ[\pi_1,\pi_3,\pi_2]\circ(\mathsf{assign}_x\times 1_{\mathbb{Z}})$

(M13) $\quad \mathsf{bc}_M = 1_{Env}\times \mathsf{bc}_S$

(M14) $\quad (\alpha)\mathsf{prop}_M = \alpha\circ(1_{Env}\times \mathsf{prop}_S)$

(M15) $\quad (\alpha_1,\alpha_2)\mathsf{rel}_M = \alpha_1\circ(\alpha_2\times 1_{\mathbb{Z}})\circ(1_{Env}\times \mathsf{rel}_S)$

(M16) $\quad (\beta)\neg_M = \beta\circ(\iota_2,\iota_1)$

(M17a) $\quad (\beta_1,\beta_2)\wedge_M = \beta_1\circ(\iota_1,\beta_2)$

(M17b) $\quad (\beta_1,\beta_2)\vee_M = \beta_1\circ(\beta_2,\iota_2)$

The Boolean expressions are treated differently from the arithmetic expressions. In the definition of $\wedge_M$, for example, β_1 can give the value false (1) and β_2 will not be evaluated, i.e., could be non-terminating: if $(e)\beta_1=<e',1>$ (false with new environment e'), then $(e)\beta_1\circ(\iota_1,\beta_2)=<e',1>$ independent of β_2.

Calling our grammar above, G, we have made $M = <M_{<st>}, M_{<ae>}, M_{<be>}>$ into a G-algebra with the seventeen definitions, (M1-M17). The algebraic semantics for G is the unique homomorphism $\theta:T_G\to M$.

5. THE TARGET LANGUAGE, T_0, THE (ENRICHED) CATEGORY OF FLOW CHARTS

Our full target language will be a category of flow charts. Morris also used flow charts for the target language but his lacked the algebraic structure that we shall describe. This algebraic structure is one of the principal advances that we have to offer; it is lacking in previous treatments of the compiler correctness problem. Further translations could be performed on the target language, and they could be proved correct.

This is because the target language has a clean algebraic character. Then the composite translation would immediately (automatically) be correct by "pasting" commuting squares together. This is our answer to our TCS referee who asks, "but who uses a compiler which generates flow charts?" Even though the flow charts are very close to machine code, we leave it to interested parties (the authors included) to carry out subsequent translations or "compilations."

The referee has also raised the question of why the category of flow charts should be so general. We will see flow charts with n "entries" and p "exits," for non-negative integers n and p. But the specification of the compiler uses only $n,p \in \{1,2\}$. This criticism is more difficult to answer. We could argue that if our language employed a conventional "**case**"-statement, then the compiler specification would use all non-negative integers n and p. But the reason is really deeper than that; our category of flow charts will have semantics in what is a well known algebraic system, an "algebraic theory" in the sense of Lawvere (1963). In an algebraic theory you can take two morphisms, one from n to p and one from q to r, take their sum and get a morphism from n+q to p+r. And this corresponds to a natural operation on flow charts, their "parallel composition." Closure under this operation demands that we have all "n-entry, p-exit" charts.

We will begin with a general description of the category of flow charts (arbitrary operation symbols) and then, later in this section, specialize to the particular operation symbols for operations on stacks and stores. Our definitions of the the flow charts and the operations on them are detailed and (we hope) complete. Accompanying each formal definition is an informal description which should be adequate for a first reading of the paper.

So to continue, let Ω be an arbitrary *one-sorted* signature or operator domain, i.e. an indexed family of disjoint sets, $<\Omega_i>_{i\in\omega}$. Viewing Ω as the union of the Ω_i, we associate with the operator domain a ranking function, $r_\Omega:\Omega\to\omega$ where $(\sigma)r_\Omega = k$ iff $\sigma\in\Omega_k$. $\Omega_\perp$ is the operator domain Ω with $\perp$ adjoined as a symbol of rank zero, i.e., $(\Omega_\perp)_0=\Omega_0\cup\{\perp\}$. Below we will fix on a specific operator domain Ω for our language T_0.

We now define flow charts, identity charts, and the operations of composition, pairing and iteration on flow charts. That these are the essential operations on charts is a key contribution of Elgot (1973). We obtain an enriched category of flow charts which is small (a set of objects instead of a proper class) by using the various [n], $n\in\omega$, as the sets of vertices. Elgot (1977) and Elgot and Shepherdson (1977) define an equivalent large category and consider the skeletal small category determined by isomorphism classes of flow charts.

In addition to the operations used in Section 4 (composition, pairing, iteration, etc.), we need the following: $0_A:[0]\to A$ is the unique function from $[0]=\emptyset$ to A; and, where A^* is the (underlying set of the) free monoid generated by a set A and $f:A\to B$ is a (total) function, $f^*:A^*\to B^*$ is the "extension" of f which takes a string $a_1...a_n$ to $(a_1)f...(a_n)f$.

Definition 5.1. A (*normalized*) $\Omega_\perp$*-flow chart from* n *to* p *of weight* s consists of a triple $<b,\tau,\ell>$ where:

begin function	$b:[n]\to[s+p]$
underlying graph	$\tau:[s]\to[s+p]^*$
labeling function	$\ell:[s]\to\Omega_\perp$,

satisfying the requirement that $|(i)\tau| = ((i)\ell)r_{\Omega_\perp}$.

(i)b is called a *begin vertex*, $i \in [s]$ is an *internal vertex*, $i \in s+[p] = \{s+j \mid j \in [p]\}$ is an *exit* and in particular, $s+j$ is the j^{th} *exit vertex*. (i)ℓ is the operation symbol labeling the i^{th} internal vertex; by the above requirement it must have rank $|(i)\tau|$. Note that the exit vertices are *not* labeled, though the begin vertices are. This makes composition of flow charts work well. Let $\mathrm{Flo}_{\Omega_\perp}(n,p)$ be the set of $\Omega_\perp$-flow charts from n to p. □

This definition of flow chart employs the convenient definition of directed ordered graph introduced by Arbib and Giveon (1968). To relate to more familiar notions of flow charts, say the function $\tau:[s] \to [s+p]^*$ takes $k \in [s]$ to $k_1 \ldots k_u \in [s+p]^*$. This says that there is an edge from vertex k to *each* of the vertices k_i ($i \in [u]$) and the natural ordering on [u] induces the (local) ordering on the edges leaving vertex k. This ordering is essential to distinguish between, for example, the "true" and "false" branches of a (binary) test node.

Definition 5.2. The *identity* $\Omega_\perp$-flow chart from n to n, denoted 1_n, has weight 0 and:

begin function	$1_{[n]}:[n] \to [n]$
underlying graph	$0_{[n]}\bullet:[0] \to [n]^*$
labeling function	$0_{\Omega_\perp}:[0] \to \Omega_\perp$.

□

Informally the identity chart from n to n has n begin vertices which are also exits and thus there is no labeling.

Definition 5.3. The *composite* of $\Omega_\perp$-flow charts, $F = \langle b,\tau,\ell \rangle$ from n to p of weight s and $F' = \langle b',\tau',\ell' \rangle$ from p to q of weight s' is $F \circ F'$ from n to q of weight $s+s'$ with:

begin function	$b \circ f:[n] \to [s+s'+q]$
underlying graph	$(\tau \circ f^*, \tau' \circ g^*):[s+s'] \to [s+s'+q]^*$
labeling function	$(\ell,\ell'):[s+s'] \to \Omega_\perp$

where f and g are the following functions,

$$f = 1_{[s]} + b':[s+p] \to [s+s'+q]$$
$$g = 0_{[s]} + 1_{[s'+q]}:[s'+q] \to [s+s'+q].$$

□

Informally $F \circ F'$ is obtained by "laying down" F and F' "end-to-end" and by identifying the p exits of F with the p begin vertices of F'. Note that the labeling works here; the labels of the identified vertices are those of F' since the exit vertices of F are not labeled. At the same time the vertices of F' are "translated" (renumbered) by adding s, i.e., a vertex j of F' becomes $s+j$ in $F \circ F'$.

Theorem 5.4. For each $n,p \in \omega$, let $\mathbf{Flo}_{\Omega_\perp}(n,p)$ be the set of $\Omega_\perp$-flow charts from n to p (i.e., $\mathrm{Flo}_{\Omega_\perp}(n,p)$). Then $\mathbf{Flo}_{\Omega_\perp}$ is a category with the nonnegative integers as objects, with composition given by Definition 5.3, and with identities given by Definition 5.2. □

Without identifying it as such, Elgot (1973) describes a category of *normal descriptions over* Ω which is essentially the same as $\mathbf{Flo}_{\Omega_\perp}$, and it is also equipped with the operations of pairing and iteration which we now proceed to define.

Definition 5.5. The *pairing* or *coalesced sum* of two $\Omega_\perp$-flow charts $F=\langle b,\tau,\ell\rangle$ from n to p of weight s and $F'=\langle b',\tau',\ell'\rangle$ from n′ to p of weight s′ is (F,F′) from n+n′ to p of weight s+s′ where

begin function	$(b\circ f, b'\circ g):[n+n']\to[s+s'+p]$
underlying graph	$(\tau\circ f^*,\tau'\circ g^*):[s+s']\to[s+s'+p]^*$
labeling function	$(\ell,\ell'):[s+s']\to\Omega_\perp$

where

$$f=1_{[s]}+0_{[s']}+1_{[p]}:[s+p]\to[s+s'+p]$$
$$g=0_{[s]}+1_{[s'+p]}:[s'+p]\to[s+s'+p].$$ □

Informally, the effect of pairing is to put the two charts F and F′ next to each other identifying the p exits of F with those of F′.

Proposition 5.6. Pairing of $\Omega_\perp$-flow charts is associative, i.e.,

$$(F_1,(F_2,F_3))=((F_1,F_2),F_3)$$

for F_1, F_2, F_3 where the pairing is defined. □

Definition 5.7. For any function $f:[n]\to[p]$ we define an associated $\Omega_\perp$-flow chart $f^\wedge$ from n to p of weight 0; $f^\wedge=\langle f,0_{[p]^*},0_{\Omega_\perp}\rangle$. □

The charts $f^\wedge$ are trivial ones which simply allow us to permute or identify exits by composition on the right; we already have an example which is the identity chart, $1_n = 1_{[n]}{}^\wedge$. Using these trivial charts corresponding to maps (Definition 5.7) and coalesced sum or pairing (Definition 5.5), we define *the separated sum* of F_i from n_i to m_i $(i\in[2])$ to be the chart

$$F_1\oplus F_2 = (F_1\circ f_1{}^\wedge, F_2\circ f_2{}^\wedge)$$

where $f_i:[s_i+m_i]\to[s_1+s_2+m_1+m_2]$ are the obvious injections for i = 1,2. Informally $F_1\oplus F_2$ is the result of laying the two charts side-by-side as is the case with pairing, except here there is no identification of exit vertices.

We want special notation for the flow charts corresponding to certain maps (injections); this is notation used for the corresponding morphisms in algebraic theories. First, $x_{(i)}^{n_1+\ldots+n_r}:n_i\to n_1+\ldots+n_r$ is $f^\wedge$, where

$$f:[n_i]\to[n_1+\ldots+n_r]$$

is the injection sending $j\in[n_i]$ to $n_1+\ldots+n_{i-1}+j$. Next (actually a special case) $x_i^n:1\to n$ is $f^\wedge$ where $f:[1]\to[n]$ sends 1 to i. In general we will not distinguish between the maps (f, above) and the corresponding charts, $x_{(i)}^{n_1+\ldots+n_r}$ and x_i^n.

The last operation is perhaps the most important operation; it is the only one that employs '⊥'. Thus all the definitions above apply to Ω-flow charts with arbitrary Ω replacing our special $\Omega_\perp$. The idea is that for an $\Omega_\perp$-flow chart from n to n+p of weight s, the 'iterate' of F, denoted $F^\dagger$, identifies the ith exit with the ith begin node, for i=1,...,n, thus introducing 'loops;' the result has p exits and weight s. The construction is more complicated than that, however, because the ith begin might be the ith exit and this iteration

(identification) has to yield an nonterminating loop ($\perp$). Worse, the first begin could be the second exit, and the second begin, the first exit; again the iteration yields non-termination. In general there could a loop of length n from the ith begin back to the ith begin in the manner indicated and the definition below finds such nodes and labels them $\perp$.

Definition 5.8. Let $F=\langle b,\tau,\ell\rangle$ be a $\Omega_\perp$-flow chart from n to n+p of weight s. Further, let $f=(x_{(1)}^{s+n+p},b,x_{(3)}^{s+n+p}):[s+n+p]\to[s+n+p]$ and factor f^n to

$$f^n = h\circ(1_s+g+1_p):[s+n+p]\to[s+n+p],$$

where $h:[s+n+p]\to[s+u+p]$ and $g:[u]\to[n]$ and u is the smallest natural number yielding such a factorization. The *iterate* of F is the flow chart $F^\dagger$ from n to p of weight s+u with:

begin function	$b\circ h:[n]\to[s+u+p]$
underlying graph	$(\tau\circ h^*,\lambda^u)):[s+u]\to[s+u+p]^*$
labeling function	$(\ell,\perp^u):[s+u]\to\Omega_\perp$,

where $\lambda^u:[u]\to[s+u+p]^*$ sends each $i\in[u]$ to $\lambda\in[s+u+p]^*$ and $\perp^u$ sends each $i\in[u]$ to $\perp\in\Omega_\perp$.

Now we present a signature (ranked alphabet) Ω which we use to construct $\Omega_\perp$-flow charts for the target language T_0 In that alphabet we include some of the symbols from the $\{int,Bool\}$-sorted signature Σ of Section 2.

$$\Omega_1 = \{load_x,\ store_x \mid x\in X\} \cup \{switch\} \cup \bigcup_{w\in\{int\}^*}\Sigma_{int,w}$$
$$\Omega_2 = \bigcup_{w\in\{int\}^*}\Sigma_{Bool,w}$$
$$\Omega_n = \emptyset,\ n = 0,3,4,\ldots .$$

This signature determines the category $\mathbf{Flo}_{\Omega_\perp}$ of $\Omega_\perp$-flow charts via Definition 5.1 and Theorem 5.4. This *is* T_0!

Once the operations and tests (Ω) have been interpreted in a (rational or continuous) algebraic theory, the interpretation of the flow charts is uniquely determined by certain natural preservation properties. The mathematics of this interpretation is postponed to Section 7; here we provide an interpretation (it is the expected interpretation) of Ω in $\mathbf{Sum}_A$ where $A = Stk\times Env$ (stacks cross environments):

$$Stk = [\omega\to\mathbb{Z}] \qquad Env = [X\to\mathbb{Z}].$$

For any set A, $\mathbf{Sum}_A$ is the algebraic theory whose morphisms from n to p consist of all *partial* functions from $A\times[n]$ to $A\times[p]$. U_0 is $\mathbf{Sum}_{Stk\times Env}$. (See Elgot (1973) where this theory is denoted [A], or ADJ (1976b).) Composition in $\mathbf{Sum}_A$ is function composition, identities are identities from **Set**, and tupling of n functions, $f_i:A\to a\times[p]$ gives $(f_1,\ldots,f_n):[n]\to[p]$ which takes $\langle a,i\rangle$ to $(a)f_i$. For distinguished morphisms,

(S1) $$x_i^n = \iota_i^A:A\to A\times[n],$$

where ι_i^A is defined in Section 4 ($a\mapsto\langle a,i\rangle$).

Note that we have taken stacks to be infinite to make the definitions simpler. For example we will write $v_1\bullet v_2\bullet\ldots\bullet v_n\bullet\rho$ where $v_i\in\mathbb{Z}$ and $\rho\in Stk$ to denote the stack whose first n elements are $v_1,\ldots,v_n$, and whose "rest" is ρ. The usual functions are associated with stacks: $push:Stk\times\mathbb{Z}\to Stk$; and, $pop:Stk\to Stk\times\mathbb{Z}$.

(S2) $\langle\rho,v\rangle push = v\bullet\rho$

(S3) $(v\bullet\rho)pop = \langle\rho,v\rangle$.

With the identification of A with $A\times[1]$, the interpretation, $I:\Omega\to\mathbf{Sum}_A$ ($A = Stk\times Env$), is given in I1-I9 below; it assigns the expected partial function to every operation and test that can occur in a T_0-flow chart. As we mentioned above this uniquely determines the interpretation of every flow chart (Section 7).

(I1) $\langle\rho, e\rangle(load_x I) = \langle(x)e\bullet\rho, e\rangle$ For $x\in X$

(I2) $\langle v\bullet\rho, e\rangle(store_x I) = \langle\rho, e[x/v]\rangle$

(I3) $\langle v_1\bullet v_2\bullet\rho, e\rangle(switchI) = \langle v_2\bullet v_1\bullet\rho, e\rangle$

(I4) $\langle\rho, e\rangle(cI) = \langle c_S\bullet\rho, e\rangle$ For $c\in\Sigma_{int,\lambda}$

(I5) $\langle v\bullet\rho, e\rangle(aop1I) = \langle(v)aop1_S\bullet\rho, e\rangle$ For $aop1\in\Sigma_{int,int}$

(I6) $\langle v_2\bullet v_1\bullet\rho, e\rangle(aop2I) = \langle(v_1,v_2)aop2_S\bullet\rho, e\rangle$ For $aop2\in\Sigma_{int,int\ int}$

(I7) $\langle\rho, e\rangle(bcI) = \langle\langle\rho, e\rangle, bc_S\rangle$ For $bc\in\Sigma_{Bool,\lambda}$

(I8) $\langle v\bullet\rho, e\rangle(propI) = \langle\langle\rho, e\rangle, (v)prop_S\rangle$ For $prop\in\Sigma_{Bool,int}$

(I9) $\langle v_1\bullet v_2\bullet\rho, e\rangle(relI) = \langle\langle\rho, e\rangle, (v_1,v_2)rel_S\rangle$ For $rel\in\Sigma_{Bool,int\ int}$

6. THE TARGET ALGEBRA OF FLOW CHARTS, T, AND THE COMPILER

Now we 'extract' a G-algebra T from T_0 as outlined in the introduction. Take $T_{<ae>}=T_{<st>}=\mathbf{Flo}_{\Omega_\perp}(1,1)$ and $T_{<be>}=\mathbf{Flo}_{\Omega_\perp}(1,2)$, where Ω is the ranked alphabet introduced at the end of the last section. We make T into a G-algebra where G is the context-free grammar of Section 3, and we do that by defining operations on $\Omega_\perp$-flow charts corresponding to each of the seventeen productions of G. This *is* the construction of the compiler because initiality of L gives the compile function (homomorphism) $\gamma:L\to T$. In the definitions of T's operations below, F, F_1, F_2 range over $T_{<ae>}=T_{<st>}=\mathbf{Flo}_{\Omega_\perp}(1,1)$ and P, P_1, P_2 range over $T_{<be>}=\mathbf{Flo}_{\Omega_\perp}(1,2)$. Thus, for example, in T11, the operation $result_T$, is just the serial composition of two *arbitrary* single entry, single exit flowcharts F_1 and F_2. If F_1 and F_2 are the flow charts compiled from a statement and an arithmetic expression, respectively, then F_1 will leave the stack as it found it and F_2 will add a single value to the stack. This last statement is a fact that one could conclude from compiler correctness, but there is nothing like this presumed or asserted in the specification of the compiler itself.

(T1) $Continue_T = 1_1$

(T2) $(F)x:=_T = F\circ store_x$

(T3) $(P,F_1,F_2)ifthenelse_T = P\circ(F_1,F_2)$

(T4) $(F_1,F_2);_T = F_1\circ F_2$

(T5) $(P,F)whiledo_T = (P\circ(F\oplus 1_1))^\dagger$

(T6) $c_T = c$

(T7) $x_T = load_x$

(T8) $(F)aop1_T = F\circ aop1$

(T9) $(F_1,F_2)aop2_T = F_1\circ F_2\circ aop2$

(T10) $(P,F_1,F_2)cond_T = P\circ(F_1,F_2)$

(T11) $(F_1,F_2)\text{result}_T = F_1 \circ F_2$

(T12) $(F_1,F_2)\text{letx}_T = \text{load}_x \circ F_1 \circ \text{store}_x \circ F_2 \circ \text{switch} \circ \text{store}_x$

(T13) $\text{bc}_T = \text{bc}$

(T14) $(F)\text{prop}_T = F \circ \text{prop}$

(T15) $(F_1,F_2)\text{rel}_T = F_1 \circ F_2 \circ \text{rel}$

(T16) $(P)\neg_T = P \circ (x_2^2, x_1^2)$

(T17a) $(P_1,P_2)\wedge_T = P_1 \circ (P_2, x_2^2)$

(T17b) $(P_1,P_2)\vee_T = P_1 \circ (x_1^2, P_2)$

7. SEMANTICS FOR FLOW CHARTS, THE TARGET THEORY U_0

We already have defined the target theory, U_0, to be the algebraic theory $\mathbf{Sum}_{\text{Stk}\times\text{Env}}$; we need the interpretation functor. Rather than going directly from $\mathbf{Flo}_{\Omega_\perp}$ to $\mathbf{Sum}_{\text{Stk}\times\text{Env}}$ it is convenient to factor that interpretation through the continuous algebraic theory freely generated by Ω, $\mathbf{CT}_\Omega$ (c.f. ADJ 1975, 1976b, 1976c, 1977). Recall that $\mathbf{CT}_\Omega(n,p)$ consists of all n-tuples of countable partial trees on the ranked alphabet Ω and variables, $x_1,\ldots,x_p$; the composition operation is simultaneous substitution. The following is a variation of an important theorem first proved by Elgot (1973).

Theorem 7.1. There is a unique functor Un (for *un*folding) from $\mathbf{Flo}_{\Omega_\perp}$ to $\mathbf{CT}_\Omega$ that preserves maps, pairing, iteration, $\perp$, and the primitives Ω. □

Theorem 7.2. (ADJ 1977) For any ω-continuous algebraic theory $\mathbf{T}$ and any interpretation $I:\Omega\to\mathbf{T}$ there exists a unique ω-continuous functor $I^\#:\mathbf{CT}_\Omega\to\mathbf{T}$ that preserves maps, pairing, iteration, $\perp$ and the interpretation (I) of the primitives Ω. □

The combination of Un from Theorem 7.1 and $I^\#$ from Theorem 7.2 (with the interpretation I of Section 5) gives us an interpretation (unique subject to certain conditions) of all Ω-flow charts; the composite $\text{Un}\circ I^\#$ goes from $\mathbf{Flo}_{\Omega_\perp}$ to $\mathbf{Sum}_{\text{Stk}\times\text{Env}}$. It is now a simple matter to describe the algebra U for the interpretation of the algebra of flow charts because each of the operations of T (Section 6) is defined in terms of operations preserved by the composite $\text{Un}\circ I^\#$.

8. THE SEMANTIC ALGEBRA FOR FLOW CHARTS, U

Take $U_{<ae>}=U_{<st>}=\mathbf{Sum}_{\text{Stk}\times\text{Env}}(1,1)$ and $U_{<be>}=\mathbf{Sum}_{\text{Stk}\times\text{Env}}(1,2)$. We make U into a G-algebra (one operation of appropriate arity for each production of G) by translating the definition of T in Section 6. This translation is possible because each of the operations used in the definitions in Section 6 (on right-hand sides) is preserved by the composite $\text{Un}\circ I^\#$. In the displayed equations defining U, the variables ϕ, ϕ_1, and ϕ_2 range over $U_{<ae>}=U_{<st>}$ while ρ, ρ_1 and ρ_2 range over $U_{<be>}$.

(U1) $\text{Continue}_U = 1_1 = 1_{\text{Stk}\times\text{Env}}$

(U2) $(\phi)x:=_U = \phi \circ (\text{store}_x I)$

(U3) $(\rho,\phi_1,\phi_2)\text{ifthenelse}_U = \rho \circ (\phi_1,\phi_2)$

(U4) $(\phi_1,\phi_2);_U = \phi_1 \circ \phi_2$

(U5) $(\rho,\phi)\text{whiledo}_U = (\rho \circ (\phi + 1_1))^\dagger$

(U6) $c_U = cI$

(U7) $x_U = \text{load}_x I$

(U8) $(\phi)\text{aop1}_U = \phi \circ (\text{aop1I})$

(U9) $(\phi_1,\phi_2)\text{aop2}_U = \phi_1 \circ \phi_2 \circ (\text{aop2I})$

(U10) $(\rho,\phi_1,\phi_2)\text{cond}_U = \rho \circ (\phi_1,\phi_2)$

(U11) $(\phi_1,\phi_2)\text{result}_U = \phi_1 \circ \phi_2$

(U12) $(\phi_1,\phi_2)\text{letx}_U = (\text{load}_x I) \circ \phi_1 \circ (\text{store}_x I). \circ \phi_2 \circ (\text{switchI}) \circ (\text{store}_x I)$

(U13) $bc_U = bcI$

(U14) $(\phi)\text{prop}_U = \phi \circ (\text{propI})$

(U15) $(\phi_1,\phi_2)\text{rel}_U = \phi_1 \circ \phi_2 \circ (\text{relI})$

(U16) $(\rho)\neg_U = \rho \circ (x_2^2, x_1^2)$

(U17a) $(\rho_1,\rho_2)\wedge_U = \rho_1 \circ (\rho_2, x_2^2)$

(U17b) $(\rho_1,\rho_2)\vee_U = \rho_1 \circ (x_1^2, \rho_2)$

Let ψ be the restriction of the composite $\text{Un} \circ I^\#$ to the carriers of T. Then ψ is a G-homomorphism because of the way U was defined (and the preservation properties of $\text{Un} \circ I^\#$) which gives algebraic semantics to the algebra T of flow charts.

9. THE ENCODING FROM PROGRAM MEANINGS TO FLOW CHART MEANINGS

As the final step before the proof of the correctness of the compiler (commuting of ★) we must define the function ε from M to U. In particular we must define ε_s for $s \in \{<ae>, <st>, <be>\}$. The proof that ★ commutes then amounts to proving that ε is in fact a homomorphism. This is accomplished in the next section. We recall the types of ε:

$$\varepsilon_{<st>}: \; M_{<st>} = [\text{Env} \multimap \text{Env}] \quad \to \quad U_{<st>} = [\text{Stk} \times \text{Env} \multimap \text{Stk} \times \text{Env}]$$
$$\varepsilon_{<ae>}: \; M_{<ae>} = [\text{Env} \multimap \text{Env} \times \mathbb{Z}] \quad \to \quad U_{<ae>} = [\text{Stk} \times \text{Env} \multimap \text{Stk} \times \text{Env}]$$
$$\varepsilon_{<be>}: \; M_{<be>} = [\text{Env} \multimap \text{Env} \times [2]] \quad \to \quad U_{<be>} = [\text{Stk} \times \text{Env} \multimap \text{Stk} \times \text{Env} \times [2]]$$

The definition of the bottom line is now given by the following.

(B1) $(\tau)\varepsilon_{<st>} = 1_{\text{Stk}} \times \tau$

(B2) $(\alpha)\varepsilon_{<ae>} = (1_{\text{Stk}} \times \alpha) \circ [\pi_1,\pi_3,\pi_2] \circ (\text{push} \times 1_{\text{Env}})$

(B3) $(\beta)\varepsilon_{<be>} = 1_{\text{Stk}} \times \beta.$

10. THE CORRECTNESS PROOF: ε IS A HOMOMORPHISM

To emphasize again the main point made by Morris in 1973 and, we believe, carried to fruition here, the correctness proof for the compiler (★ commutes) now reduces to seventeen little proofs or lemmas; one lemma for each operation ξ of G (Section 3). We must prove that ε is a homomorphism, i.e., that

$$((\gamma_1,\ldots,\gamma_n)\xi_M)\varepsilon = ((\gamma_1)\varepsilon,\ldots,(\gamma_n)\varepsilon)\xi_U$$

for each of the seventeen instances of ξ as given in M1-M17.

This proof process has some very intriguing aspects. The proofs of the lemmas are all equational, each line being justified by some previous line, some definition (M1-M17, U1-U17, and B1-B3) or some fact about the operations involved in those definitions. We divide these latter facts into three groups.

(E) Properties of the underlying data type.

(F) Properties of the "storage" operations (push, fetch_x, etc).

(G) Properties of the set-theoretic operators like composition, identities, tupling, sum and product.

Even though we make the advice that all properties of the underlying data type(s) be included in the specification of the language (E1-E27), we will have no need for these facts in connection with the proof of compiler correctness. Presumably program correctness and program transformation in the proposed style would use properties of this first kind.

The second kind of justification will depend on the particular kind of mathematical semantics given for the languages (source and target). In our case we must relate functions like those associated with load_x, store_x, switch, with those used in the semantics of M like fetch_x and assign_x. Each of the assertions in this group has a simple set-theoretic proof, depending, in part, on properties of the third kind (G). The first nine (F1-F9) are reformulations of the definition of the interpretation function I (I1-I9). In the latter case we chose to give "argument - value" presentations of the meanings of the flow chart primitives because such are much simpler and clearer than the alternative "closed form" presentations below. However, we can equationally manipulate these closed form characterizations, something we could not do with I1-I9. And it is the equational (algebraic) proof method that we are aiming for in the details of the correctness argument.

(F1) $\text{load}_x\text{I} = (1_{\text{Stk}}\times\text{fetch}_x)\circ[\pi_1,\pi_3,\pi_2]\circ(\text{push}\times 1_{\text{Env}})$

(F2) $\text{store}_x\text{I} = (\text{pop}\times 1_{\text{Env}})\circ[\pi_1,\pi_3,\pi_2]\circ(1_{\text{Stk}}\times\text{assign}_x)$

(F3) $\text{switchI} = (\text{pop}\times 1_{\text{Env}})\circ(\text{pop}\times 1_{\mathbb{Z}\times\text{Env}})\circ[\pi_1,\pi_3,\pi_2,\pi_4]\circ(\text{push}\times 1_{\mathbb{Z}\times\text{Env}})\circ(\text{push}\times 1_{\text{Env}})$

(F4) $\text{cI} = (1_{\text{Stk}}\times c_S\times 1_{\text{Env}})\circ(\text{push}\times 1_{\text{Env}})$

(F5) $\text{aop1I} = (\text{pop}\times 1_{\text{Env}})\circ(1_{\text{Stk}}\times\text{aop1}_S\times 1_{\text{Env}})\circ(\text{push}\times 1_{\text{Env}})$

(F6) $\text{aop2I} = (\text{pop}\times 1_{\text{Env}})\circ(\text{pop}\times 1_{\mathbb{Z}}\times 1_{\text{Env}})\circ(1_{\text{Stk}}\times\text{aop2}_S\times 1_{\text{Env}})\circ(\text{push}\times 1_{\text{Env}})$

(F7) $\text{bcI} = 1_{\text{Stk}\times\text{Env}}\times\text{bc}_S$

(F8) $\text{propI} = (\text{pop}\times 1_{\text{Env}})\circ[\pi_1,\pi_3,\pi_2]\circ(1_{\text{Stk}\times\text{Env}}\times\text{prop}_S)$

(F9) $\text{relI} = (\text{pop}\times 1_{\text{Env}})\circ(\text{pop}\times 1_{\mathbb{Z}\times\text{Env}})\circ[\pi_1,\pi_3,\pi_2,\pi_4]\circ(1_{\text{Stk}\times\text{Env}}\times\text{rel}_S)$

(FX) $\text{push}\circ\text{pop} = 1_{\text{Stk}\times\mathbb{Z}}$

(FXa) $[\pi_1,\pi_3,\pi_2]\circ(\text{push}\times 1_{\text{Env}})\circ(\text{pop}\times 1_{\text{Env}})\circ[\pi_1,\pi_3,\pi_2] = 1_{\text{Stk}\times\times\text{Env}\times\mathbb{Z}}$

The last are the most interesting properties for they are general and, in effect, category theoretic. Presumably the set of these equations is pretty small and will not keep changing with different languages or styles. This suggests the plausibility of Mosses' approach to "making denotational semantics less concrete," (Mosses (1977, 1978)).

(G0) $1_A \circ f = f = f \circ 1_B$

(G1) $(f \circ g) \circ h = f \circ (g \circ h)$

(G2) $(f \times g) \times h = f \times (g \times h)$

(G3) $1_A \times 1_B = 1_{A \times B}$

(G4) $1_A \times (f \circ g) = (1_A \times f) \circ (1_A \times g)$

(G5) $(f \times g) \circ (h \times k) = (f \circ h) \times (g \circ k)$

(G6) $(f \times 1_C) \circ (1_B \times g) = f \times g = (1_A \times g) \circ (f \times 1_D)$

(C1) $1_A \times \iota_j^B = \iota_j^{A \times B}$

(C2) $1_A \times (f,g) = (1_A \times f, 1_A \times g)$

(C3) $1_A \times (f+g) = (1_A \times f) + (1_A \times g)$

(C4) $1_A \times f^\dagger = (1_A \times f)^\dagger$

(C5) $(f,g) \circ h = (f \circ h, g \circ h)$

The following identities are necessary for permuting arguments for functions, i.e., manipulating tuples of projection functions.

(P1) $[\pi_1, \pi_2, \ldots, \pi_n] = 1$

Let q,r:[n]→[n] be permutations of [n].

(P2) $[\pi_{1q}, \pi_{2q}, \ldots, \pi_{nq}] \circ [\pi_{1r}, \pi_{2r}, \ldots, \pi_{nr}] = [\pi_{1rq}, \pi_{2rq}, \ldots, \pi_{nrq}]$

(P3) $1_A \times [\pi_{1q}, \pi_{2q}, \ldots, \pi_{nq}] = [\pi_1, \pi_{1q+1}, \pi_{2q+1}, \ldots, \pi_{nq+1}]$

For monadic functions $f_i: A_i \to B_i$ there is a convenient general rule for permuting arguments:

(P4) $(f_1 \times \ldots \times f_n) \circ [\pi_{1q}, \ldots, \pi_{nq}] = [\pi_{1q}, \ldots, \pi_{nq}] \circ (f_{1q} \times \ldots \times f_{nq})$.

But when the functions involved have cartesian products for sources and/or targets, then the corresponding scheme has a very complicated statement. Below we list the special cases of that general scheme which we will need in proofs to follow. Assume $f_i: A_i \to B_i$, $c: \to C$, $g: C_1 \times C_2 \to D$ and $h: C \to D_1 \times D_2$.

(P4a) $(f_1 \times f_2 \times c) \circ [\pi_1, \pi_3, \pi_2] = f_1 \times c \times f_2$

(P4b) $(g \times f_1 \times f_2) \circ [\pi_1, \pi_3, \pi_2] = [\pi_1, \pi_2, \pi_4, \pi_3] \circ (g \times f_2 \times f_1)$

(P4c) $(f_1 \times g \times f_2) \circ [\pi_1, \pi_3, \pi_2] = [\pi_1, \pi_4, \pi_2, \pi_3] \circ (f_1 \times f_2 \times g)$

(P4d) $(f_1 \times f_2 \times g) \circ [\pi_1, \pi_3, \pi_2] = [\pi_1, \pi_3, \pi_4, \pi_2] \circ (f_1 \times g \times f_2)$

(P4e) $[\pi_1, \pi_3, \pi_2] \circ (h \times f_1 \times f_2) = (h \times f_2 \times f_1) \circ [\pi_1, \pi_2, \pi_4, \pi_3]$

(P4f) $[\pi_1, \pi_3, \pi_2] \circ (f_1 \times h \times f_2) = (f_1 \times f_2 \times h) \circ [\pi_1, \pi_3, \pi_4, \pi_2]$

(P4g) $[\pi_1, \pi_3, \pi_2] \circ (f_1 \times f_2 \times h) = (f_1 \times h \times f_2) \circ [\pi_1, \pi_4, \pi_2, \pi_3]$

To save space in displaying the proofs we will abbreviate the isomorphism $[\pi_{1q}, \ldots, \pi_{nq}]$ with the sequence [1q...nq] which will not need commas since $n<10$ (thank goodness). In addition we will abbreviate Stk, Env and $\mathbb{Z}$ by S, E and Z respectively. Use of associativity of ∘ (G1) and of × (G2) will not be mentioned explicitly in the proofs.

Now we proceed with the 17 (actually 18 because ∧ and ∨ are treated separately) proofs. Each proof will be a line-by-line proof with justifications (on the right) coming from previous facts and definitions.

Observe the form; they begin with the definition in M, the definition of ε (B1,2,3), and then the various facts. In the middle we are justifying what at times seem to be tediously manipulative steps; this is particularly true in proofs (9), (12) and (15), and in them, in applications of (FX), (FXa) and (P4a-P4g). The proofs conclude with the definition (again) of ε and of operations in U.

(1) $(\text{continue}_M)\varepsilon_{<st>} = (1_E)\varepsilon_{<st>}$ (M1)
$= 1_S \times 1_E$ (B1)
$= 1_{S\times E}$ (G3)
$= \text{continue}_U$ (U1)

(2) $((\alpha)x:=_M)\varepsilon_{<st>} = (\alpha \circ \text{assign}_x)\varepsilon_{<st>}$ (M2)
$= 1_S \times (\alpha \circ \text{assign}_x)$ (B1)
$= (1_S \times \alpha) \circ [132] \circ [132] \circ (1_S \times \text{assign}_x)$ (G4)
$= (1_S \times \alpha) \circ [132] \circ (\text{push} \times 1_E) \circ (\text{pop} \times 1_E) \circ [132] \circ (1_S \times \text{assign}_x)$ (FXa)
$= (1_S \times \alpha) \circ [132] \circ (\text{push} \times 1_E) \circ (\text{store}_x I)$ (F2)
$= (\alpha)\varepsilon_{<ae>} \circ (\text{store}_x I)$ (B2)
$= ((\alpha)\varepsilon_{<ae>})x:=_U$ (U2)

(3) $((\beta,\tau_1,\tau_2)\text{ifthenelse}_M)\varepsilon_{<st>} = (\beta \circ (\tau_1,\tau_2))\varepsilon_{<st>}$ (M3)
$= 1_S \times (\beta \circ (\tau_1,\tau_2))$ (B1)
$= (1_S \times \beta) \circ (1_S \times (\tau_1,\tau_2))$ (G4)
$= (1_S \times \beta) \circ (1_S \times \tau_1, 1_S \times \tau_2)$ (C2)
$= (\beta)\varepsilon_{<be>} \circ ((\tau_1)\varepsilon_{<st>}, (\tau_2)\varepsilon_{<st>})$ (B1,B3)
$= ((\beta)\varepsilon_{<be>}, (\tau_1)\varepsilon_{<st>}, (\tau_2)\varepsilon_{<st>})\text{ifthenelse}_U$ (U3)

(4) $((\tau_1,\tau_2);_M)\varepsilon_{<st>} = (\tau_1 \circ \tau_2)\varepsilon_{<st>}$ (M4)
$= 1_S \times (\tau_1 \circ \tau_2)$ (B1)
$= (1_S \times \tau_1) \circ (1_S \times \tau_2)$ (G4)
$= (\tau_1)\varepsilon_{<st>} \circ (\tau_2)\varepsilon_{<st>}$ (B1)
$= ((\tau_1)\varepsilon_{<st>}, (\tau_2)\varepsilon_{<st>});_U$ (U4)

(5) $((\beta,\tau)\text{whiledo}_M)\varepsilon_{<st>} = ((\beta \circ (\tau + 1_E))^\dagger)\varepsilon_{<st>}$ (M5)
$= 1_S \times (\beta \circ (\tau + 1_E))^\dagger$ (B1)
$= (1_S \times (\beta \circ (\tau + 1_E)))^\dagger$ (C4)
$= ((1_S \times \beta) \circ (1_S \times (\tau + 1_E)))^\dagger$ (G4)
$= ((1_S \times \beta) \circ ((1_S \times \tau) + (1_S \times 1_E)))^\dagger$ (C3)
$= ((1_S \times \beta) \circ ((1_S \times \tau) + 1_{S\times E}))^\dagger$ (G3)
$= ((\beta)\varepsilon_{<be>} \circ ((\tau)\varepsilon_{<st>} + 1_{E\times S}))^\dagger$ (B1,3)
$= ((\beta)\varepsilon_{<be>} \circ ((\tau)\varepsilon_{<st>} + 1_1))^\dagger$ (U1)
$= ((\beta)\varepsilon_{<be>}, (\tau)\varepsilon_{<st>})\text{whiledo}_U$ (U5)

(6) $(c_M)\varepsilon_{<ae>} = (1_E \times c_S)\varepsilon_{<ae>}$ (M6)
$= (1_S \times 1_E \times c_S) \circ [132] \circ (\text{push} \times 1_E)$ (B2)
$= (1_S \times c_S \times 1_E) \circ (\text{push} \times 1_E)$ (P4a)

$= cI$ (F3)

$= c_U$ (U6)

(7) $(x_M)\varepsilon_{<ae>} = (\text{fetch}_x)\varepsilon_{<ae>}$ (M7)

$= (1_S \times \text{fetch}_x) \circ [132] \circ (\text{push} \times 1_E)$ (B2)

$= \text{load}_x I$ (F1)

$= x_M$ (U7)

(8) $((\alpha)\text{aop1}_M)\varepsilon_{<ae>} = (\alpha \circ (1_E \times \text{aop1}_S))\varepsilon_{<ae>}$ (M8)

$= (1_S \times (\alpha \circ (1_E \times \text{aop1}_S))) \circ [132] \circ (\text{push} \times 1_E)$ (B2)

$= (1_S \times \alpha) \circ (1_S \times 1_E \times \text{aop1}_S) \circ [132] \circ (\text{push} \times 1_E)$ (G4)

$= (1_S \times \alpha) \circ [132] \circ (1_S \times \text{aop1}_S \times 1_E) \circ (\text{push} \times 1_E)$ (P4)

$= (1_S \times \alpha) \circ [132] \circ (\text{push} \times 1_E) \circ (\text{pop} \times 1_E) \circ (1_S \times \text{aop1}_S \times 1_E) \circ (\text{push} \times 1_E)$ (F2)

$= (\alpha)\varepsilon_{<ae>} \circ (\text{pop} \times 1_E) \circ (1_S \times \text{aop1}_S \times 1_E) \circ (\text{push} \times 1_E)$ (B2)

$= (\alpha)\varepsilon_{<ae>} \circ (\text{aop1I})$ (F6)

$= ((\alpha)\varepsilon_{<ae>})\text{aop1}_U$ (U8)

(9) $((\alpha_1,\alpha_2)\text{aop2}_M)\varepsilon_{<ae>} = (\alpha_1 \circ (\alpha_2 \times 1_Z) \circ [132] \circ (1_E \times \text{aop2}_S))\varepsilon_{<ae>}$ (M9)

$= (1_S \times (\alpha_1 \circ (\alpha_2 \times 1_Z) \circ [132] \circ (1_E \times \text{aop2}_S))) \circ [132] \circ (\text{push} \times 1_E)$ (B2)

$= (1_S \times (\alpha_1 \circ (\alpha_2 \times 1_Z))) \circ (1_S \times [132]) \circ (1_S \times 1_E \times \text{aop2}_S) \circ [132] \circ (\text{push} \times 1_E)$ (G4)

$= (1_S \times (\alpha_1 \circ (\alpha_2 \times 1_Z))) \circ [1243] \circ (1_S \times 1_E \times \text{aop2}_S) \circ [132] \circ (\text{push} \times 1_E)$ (P3)

$= (1_S \times (\alpha_1 \circ (\alpha_2 \times 1_Z))) \circ [1243] \circ [1342] \circ (1_S \times \text{aop2}_S \times 1_E) \circ (\text{push} \times 1_E)$ (P4d)

$= (1_S \times (\alpha_1 \circ (\alpha_2 \times 1_Z))) \circ [1432] \circ (1_S \times \text{aop2}_S \times 1_E) \circ (\text{push} \times 1_E)$ (P2)

$= (1_S \times (\alpha_1 \circ (\alpha_2 \times 1_Z))) \circ [1432] \circ (\text{push} \times 1_Z \times 1_E) \circ (\text{pop} \times 1_Z \times 1_E) \circ (1_S \times \text{aop2}_S \times 1_E) \circ (\text{push} \times 1_E)$ (FX)

$= (1_S \times (\alpha_1 \circ (\alpha_2 \times 1_Z))) \circ [1432] \circ (\text{push} \times 1_Z \times 1_E) \circ (\text{push} \times 1_E) \circ (\text{pop} \times 1_E) \circ$
$(\text{pop} \times 1_Z \times 1_E) \circ (1_S \times \text{aop2}_S \times 1_E) \circ (\text{push} \times 1_E)$ (FX)

$= (1_S \times (\alpha_1 \circ (\alpha_2 \times 1_Z))) \circ [1432] \circ (\text{push} \times 1_Z \times 1_E) \circ (\text{push} \times 1_E) \circ (\text{aop2I})$ (F6)

$= (1_S \times \alpha_1) \circ (1_S \times \alpha_2 \times 1_Z) \circ [1432] \circ (\text{push} \times 1_Z \times 1_E) \circ (\text{push} \times 1_E) \circ (\text{aop2I})$ (G4)

$= (1_S \times \alpha_1) \circ (1_S \times \alpha_2 \times 1_Z) \circ [1423] \circ [1243] \circ (\text{push} \times 1_Z \times 1_E) \circ (\text{push} \times 1_E) \circ (\text{aop2I})$ (P2)

$= (1_S \times \alpha_1) \circ [132] \circ (1_S \times 1_Z \times \alpha_2) \circ [1243] \circ (\text{push} \times 1_Z \times 1_E) \circ (\text{push} \times 1_E) \circ (\text{aop2I})$ (P4g)

$= (1_S \times \alpha_1) \circ [132] \circ (1_S \times 1_Z \times \alpha_2) \circ (\text{push} \times 1_E \times 1_Z) \circ [132] \circ (\text{push} \times 1_E) \circ (\text{aop2I})$ (P4b)

$= (1_S \times \alpha_1) \circ [132] \circ (\text{push} \times 1_E) \circ (1_S \times \alpha_2) \circ [132] \circ (\text{push} \times 1_E) \circ (\text{aop2I})$ (G6)

$= ((\alpha_1)\varepsilon_{<ae>}) \circ ((\alpha_2)\varepsilon_{<ae>}) \circ (\text{aop2I})$ (B2)

$= ((\alpha_1)\varepsilon_{<ae>},(\alpha_2)\varepsilon_{<ae>})\text{aop2}_U$ (U9)

(10) $((\beta,\alpha_1,\alpha_2)\text{cond}_M)\varepsilon_{<ae>} = (\beta \circ (\alpha_1,\alpha_2))\varepsilon_{<ae>}$ (M10)

$= (1_S \times (\beta \circ (\alpha_1,\alpha_2))) \circ [132] \circ (\text{push} \times 1_E)$ (B2)

$= (1_S \times \beta) \circ (1_S \times (\alpha_1,\alpha_2)) \circ [132] \circ (\text{push} \times 1_E)$ (G4)

$= (1_S \times \beta) \circ (1_S \times \alpha_1, 1_S \times \alpha_2) \circ [132] \circ (\text{push} \times 1_E)$ (C2)

$= (1_S \times \beta) \circ ((1_S \times \alpha_1) \circ [132] \circ (\text{push} \times 1_E), (1_S \times \alpha_2) \circ [132] \circ (\text{push} \times 1_E))$ (C5)

$= (\beta)\varepsilon_{<be>} \circ ((\alpha_1)\varepsilon_{<ae>},(\alpha_2)\varepsilon_{<ae>})$ (B2,B3)

$= ((\beta)\varepsilon_{<be>},(\alpha_1)\varepsilon_{<ae>},(\alpha_2)\varepsilon_{<ae>})\text{cond}_U$ (U10)

(11) $(\tau,\alpha)\text{result}_M = (\tau \circ \alpha)\varepsilon_{<ae>}$ (M11)

$= (1_S \times (\tau \circ \alpha)) \circ [132] \circ (\text{push} \times 1_E)$ (B2)

$$= (1_S \times \tau) \circ (1_S \times \alpha) \circ [132] \circ (\mathrm{push} \times 1_E) \qquad \text{(G4)}$$

$$= (\tau)\varepsilon_{<st>} \circ (\alpha)\varepsilon_{<ae>} \qquad \text{(B1,B2)}$$

$$= ((\tau)\varepsilon_{<st>}, (\alpha)\varepsilon_{<ae>})\mathrm{result}_U \qquad \text{(U11)}$$

(12) $((\alpha_1,\alpha_2)\mathrm{let}x_M)\varepsilon_{<ae>}$

$$= (\mathrm{fetch}_x \circ ((\alpha_1 \circ \mathrm{assign}_x \circ \alpha_2) \times 1_Z) \circ [132] \circ (\mathrm{assign}_x \times 1_Z))\varepsilon_{<ae>} \qquad \text{(M12)}$$

$$= 1_S \times (\mathrm{fetch}_x \circ ((\alpha_1 \circ \mathrm{assign}_x \circ \alpha_2) \times 1_Z) \circ [132] \circ (\mathrm{assign}_x \times 1_Z)) \circ [132] \circ (\mathrm{push} \times 1_E) \qquad \text{(B2)}$$

$$= (1_S \times \mathrm{fetch}_x) \circ (1_S \times (\alpha_1 \circ \mathrm{assign}_x \circ \alpha_2) \times 1_Z) \circ [1243] \circ (1_S \times \mathrm{assign}_x \times 1_Z)) \circ [132] \circ (\mathrm{push} \times 1_E) \qquad \text{(G4,P3)}$$

$$= (1_S \times \mathrm{fetch}_x) \circ (1_S \times (\alpha_1 \circ \mathrm{assign}_x \circ \alpha_2) \times 1_Z) \circ [1243] \circ [1423] \circ (1_{S\times Z} \times \mathrm{assign}_x) \circ (\mathrm{push} \times 1_E) \qquad \text{(P4c,G3)}$$

$$= (1_S \times \mathrm{fetch}_x) \circ (1_S \times (\alpha_1 \circ \mathrm{assign}_x \circ \alpha_2) \times 1_Z) \circ [1324] \circ (1_{S\times Z} \times \mathrm{assign}_x) \circ (\mathrm{push} \times 1_E) \qquad \text{(P2)}$$

$$= (1_S \times \mathrm{fetch}_x) \circ (1_S \times (\alpha_1 \circ \mathrm{assign}_x \circ \alpha_2) \times 1_Z) \circ [1324] \circ (\mathrm{push} \times 1_{E\times Z}) \circ (1_S \times \mathrm{assign}_x) \qquad \text{(G6)}$$

$$= (1_S \times \mathrm{fetch}_x) \circ (1_S \times (\alpha_1 \circ \mathrm{assign}_x \circ \alpha_2) \times 1_Z) \circ [1324] \circ (\mathrm{push} \times 1_{E\times Z}) \circ [132] \circ (\mathrm{push} \times 1_E) \circ (\mathrm{pop} \times 1_E) \circ [132] \circ (1_S \times \mathrm{assign}_x) \qquad \text{(G0,FXa)}$$

$$= (1_S \times \mathrm{fetch}_x) \circ (1_S \times (\alpha_1 \circ \mathrm{assign}_x \circ \alpha_2) \times 1_Z) \circ [1324] \circ (\mathrm{push} \times 1_{E\times Z}) \circ [132] \circ (\mathrm{push} \times 1_E) \circ (\mathrm{store}_x I) \qquad \text{(F2)}$$

$$= (1_S \times \mathrm{fetch}_x) \circ (1_S \times (\alpha_1 \circ \mathrm{assign}_x \circ \alpha_2) \times 1_Z) \circ [1324] \circ [1243] \circ (\mathrm{push} \times 1_{Z\times E}) \circ (\mathrm{push} \times 1_E) \circ (\mathrm{store}_x I) \qquad \text{(G3,P4b)}$$

$$= (1_S \times \mathrm{fetch}_x) \circ (1_S \times (\alpha_1 \circ \mathrm{assign}_x \circ \alpha_2) \times 1_Z) \circ [1342] \circ (\mathrm{push} \times 1_{Z\times E}) \circ (\mathrm{push} \times 1_E) \circ (\mathrm{store}_x I) \qquad \text{(P2)}$$

$$= (1_S \times \mathrm{fetch}_x) \circ [132] \circ (\mathrm{push} \times 1_E) \circ (\mathrm{pop} \times 1_E) \circ [132] \circ (1_S \times (\alpha_1 \circ \mathrm{assign}_x \circ \alpha_2) \times 1_Z) \circ [1342] \circ (\mathrm{push} \times 1_{Z\times E}) \circ (\mathrm{push} \times 1_E) \circ (\mathrm{store}_x I) \qquad \text{(G0,FXa)}$$

$$= (\mathrm{load}_x I) \circ (\mathrm{pop} \times 1_E) \circ [132] \circ (1_S \times (\alpha_1 \circ \mathrm{assign}_x \circ \alpha_2) \times 1_Z) \circ [1342] \circ (\mathrm{push} \times 1_{Z\times E}) \circ (\mathrm{push} \times 1_E) \circ (\mathrm{store}_x I) \qquad \text{(F1)}$$

$$= (\mathrm{load}_x I) \circ (\mathrm{pop} \times 1_E) \circ (1_{S\times Z} \times (\alpha_1 \circ \mathrm{assign}_x \circ \alpha_2)) \circ [1342] \circ [1342] \circ (\mathrm{push} \times 1_{Z\times E}) \circ (\mathrm{push} \times 1_E) \circ (\mathrm{store}_x I) \qquad \text{(P4f)}$$

$$= (\mathrm{load}_x I) \circ (\mathrm{pop} \times 1_E) \circ (1_{S\times Z} \times (\alpha_1 \circ \mathrm{assign}_x \circ \alpha_2)) \circ [1423] \circ (\mathrm{push} \times 1_{Z\times E}) \circ (\mathrm{push} \times 1_E) \circ (\mathrm{store}_x I) \qquad \text{(P2,P1)}$$

$$= (\mathrm{load}_x I) \circ (1_S \times (\alpha_1 \circ \mathrm{assign}_x \circ \alpha_2)) \circ (\mathrm{pop} \times 1_{E\times Z}) \circ [1423] \circ (\mathrm{push} \times 1_{Z\times E}) \circ (\mathrm{push} \times 1_E) \circ (\mathrm{store}_x I) \qquad \text{(G6)}$$

$$= (\mathrm{load}_x I) \circ (1_S \times \alpha_1) \circ (1_S \times \mathrm{assign}_x) \circ (1_S \times \alpha_2) \circ (\mathrm{pop} \times 1_{E\times Z}) \circ [1423] \circ (\mathrm{push} \times 1_{Z\times E}) \circ (\mathrm{push} \times 1_E) \circ (\mathrm{store}_x I) \qquad \text{(G4)}$$

$$= (\mathrm{load}_x I) \circ (1_S \times \alpha_1) \circ [132] \circ (\mathrm{push} \times 1_E) \circ (\mathrm{pop} \times 1_E) \circ [132] \circ (1_S \times \mathrm{assign}_x) \circ (1_S \times \alpha_2) \circ (\mathrm{pop} \times 1_{E\times Z}) \circ [1423] \circ (\mathrm{push} \times 1_{Z\times E}) \circ (\mathrm{push} \times 1_E) \circ (\mathrm{store}_x I) \qquad \text{(FXa)}$$

$$= (\mathrm{load}_x I) \circ (\alpha_1)\varepsilon_{<ae>} \circ (\mathrm{store}_x I) \circ (1_S \times \alpha_2) \circ (\mathrm{pop} \times 1_{E\times Z}) \circ [1423] \circ (\mathrm{push} \times 1_{Z\times E}) \circ (\mathrm{push} \times 1_E) \circ (\mathrm{store}_x I) \qquad \text{(B2,F2)}$$

$$= (\mathrm{load}_x I) \circ (\alpha_1)\varepsilon_{<ae>} \circ (\mathrm{store}_x I) \circ (1_S \times \alpha_2) \circ [132] \circ (\mathrm{push} \times 1_E) \circ (\mathrm{pop} \times 1_E) \circ [132] \circ (\mathrm{pop} \times 1_{E\times Z}) \circ [1423] \circ (\mathrm{push} \times 1_{Z\times E}) \circ (\mathrm{push} \times 1_E) \circ (\mathrm{store}_x I) \qquad \text{(FXa)}$$

$$= (\mathrm{load}_x I) \circ (\alpha_1)\varepsilon_{<ae>} \circ (\mathrm{store}_x I) \circ (\alpha_2)\varepsilon_{<ae>} \circ (\mathrm{pop} \times 1_E) \circ [132] \circ (\mathrm{pop} \times 1_{E\times Z}) \circ [1423] \circ (\mathrm{push} \times 1_{Z\times E}) \circ (\mathrm{push} \times 1_E) \circ (\mathrm{store}_x I) \qquad \text{(B2)}$$

$$= (\mathrm{load}_x I) \circ (\alpha_1)\varepsilon_{<ae>} \circ (\mathrm{store}_x I) \circ (\alpha_2)\varepsilon_{<ae>} \circ (\mathrm{pop} \times 1_E) \circ (\mathrm{pop} \times 1_{Z\times E}) \circ [1243] \circ [1423] \circ (\mathrm{push} \times 1_{Z\times E}) \circ (\mathrm{push} \times 1_E) \circ (\mathrm{store}_x I) \qquad \text{(P4e)}$$

$$= (\mathrm{load}_x\mathrm{I}) \circ (\alpha_1)\varepsilon_{<ae>} \circ (\mathrm{store}_x\mathrm{I}) \circ (\alpha_2)\varepsilon_{<ae>} \circ (\mathrm{pop} \times 1_E) \circ$$
$$(\mathrm{pop} \times 1_{Z \times E}) \circ [1324] \circ (\mathrm{push} \times 1_{Z \times E}) \circ (\mathrm{push} \times 1_E) \circ (\mathrm{store}_x\mathrm{I}) \quad \text{(P2)}$$
$$= (\mathrm{load}_x\mathrm{I}) \circ (\alpha_1)\varepsilon_{<ae>} \circ (\mathrm{store}_x\mathrm{I}) \circ (\alpha_2)\varepsilon_{<ae>} \circ (\mathrm{switchI}) \circ (\mathrm{store}_x\mathrm{I}) \quad \text{(F3)}$$
$$= ((\alpha_1)\varepsilon_{<ae>},(\alpha_2)\varepsilon_{<ae>})\mathrm{let}x_U \quad \text{(U12)}$$

(13) $(\mathrm{bc}_M)\varepsilon_{<be>} = (1_E \times \mathrm{bc}_S)\varepsilon_{<be>}$ (M13)
$$= 1_S \times 1_E \times \mathrm{bc}_S \quad \text{(B3)}$$
$$= 1_{S \times E} \times \mathrm{bc}_S \quad \text{(G3)}$$
$$= \mathrm{bcI} \quad \text{(F7)}$$
$$= \mathrm{bc}_U \quad \text{(U13)}$$

(14) $((\alpha)\mathrm{prop}_M)\varepsilon_{<be>} = (\alpha \circ (1_E \circ \mathrm{prop}_S))\varepsilon_{<be>}$ (M14)
$$= 1_S \times (\alpha \circ (1_E \times \mathrm{prop}_S)) \quad \text{(B3)}$$
$$= (1_S \times 1\alpha) \circ (1_S \times 1_E \times \mathrm{prop}_S) \quad \text{(G4)}$$
$$= (1_S \times 1\alpha) \circ [132] \circ [132] \circ (1_S \times 1_E \times \mathrm{prop}_S) \quad \text{(P2,P1)}$$
$$= (1_S \times 1\alpha) \circ [132] \circ (\mathrm{push} \times 1_E) \circ (\mathrm{pop} \times 1_E) \circ [132] \circ (1_S \times 1_E \times \mathrm{prop}_S) \quad \text{(FXa)}$$
$$= (\alpha)\varepsilon_{<ae>} \circ (\mathrm{pop} \times 1_E) \circ [132] \circ (1_S \times 1_E \times \mathrm{prop}_S) \quad \text{(B2)}$$
$$= (\alpha)\varepsilon_{<ae>} \circ (\mathrm{propI}) \quad \text{(F8)}$$
$$= ((\alpha)\varepsilon_{<ae>})\mathrm{prop}_U \quad \text{(U14)}$$

(15) $((\alpha_1,\alpha_2)\mathrm{rel}_M)\varepsilon_{<be>} = (\alpha_1 \circ (\alpha_2 \times 1_Z) \circ [132] \circ (1_E \times \mathrm{rel}_S))\varepsilon_{<be>}$ (M15)
$$= 1_S \times (\alpha_1 \circ (\alpha_2 \times 1_Z) \circ [132] \circ (1_E \times \mathrm{rel}_S)) \quad \text{(B2)}$$
$$= (1_S \times (\alpha_1 \circ (\alpha_2 \times 1_Z))) \circ (1_S \times [132]) \circ (1_{S \times E} \times \mathrm{rel}_S) \quad \text{(G4)}$$
$$= (1_S \times (\alpha_1 \circ (\alpha_2 \times 1_Z))) \circ [1243] \circ (1_{S \times E} \times \mathrm{rel}_S) \quad \text{(P3)}$$
$$= (1_S \times \alpha_1) \circ (1_S \times \alpha_2 \times 1_Z) \circ [1243] \circ (1_{S \times E} \times \mathrm{rel}_S) \quad \text{(G4)}$$
$$= (1_S \times \alpha_1) \circ [132] \circ (\mathrm{push} \times 1_E) \circ (\mathrm{pop} \times 1_E) \circ [132] \circ (1_S \times \alpha_2 \times 1_Z) \circ [1243] \circ (1_{S \times E} \times \mathrm{rel}_S) \quad \text{(FXa)}$$
$$= (\alpha_1)\varepsilon_{<ae>} \circ (\mathrm{pop} \times 1_E) \circ [132] \circ (1_S \times \alpha_2 \times 1_Z) \circ [1243] \circ (1_{S \times E} \times \mathrm{rel}_S) \quad \text{(B2)}$$
$$= (\alpha_1)\varepsilon_{<ae>} \circ (\mathrm{pop} \times 1_E) \circ (1_{S \times Z} \times \alpha_2) \circ [1342] \circ [1243] \circ (1_{S \times E} \times \mathrm{rel}_S) \quad \text{(P4f)}$$
$$= (\alpha_1)\varepsilon_{<ae>} \circ (\mathrm{pop} \times 1_E) \circ (1_{S \times Z} \times \alpha_2) \circ [1324] \circ (1_{S \times E} \times \mathrm{rel}_S) \quad \text{(P2)}$$
$$= (\alpha_1)\varepsilon_{<ae>} \circ (1_S \times \alpha_2) \circ (\mathrm{pop} \times 1_{E \times Z}) \circ [1324] \circ (1_{S \times E} \times \mathrm{rel}_S) \quad \text{(G6)}$$
$$= (\alpha_1)\varepsilon_{<ae>} \circ (1_S \times \alpha_2) \circ [132] \circ (\mathrm{push} \times 1_E) \circ (\mathrm{pop} \times 1_E) \circ [132] \circ (\mathrm{pop} \times 1_{E \times Z}) \circ [1324] \circ (1_{S \times E} \times \mathrm{rel}_S) \quad \text{(FXa)}$$
$$= (\alpha_1)\varepsilon_{<ae>} \circ (\alpha_2)\varepsilon_{<ae>} \circ (\mathrm{pop} \times 1_E) \circ [132] \circ (\mathrm{pop} \times 1_{E \times Z}) \circ [1324] \circ (1_{S \times E} \times \mathrm{rel}_S) \quad \text{(B2)}$$
$$= (\alpha_1)\varepsilon_{<ae>} \circ (\alpha_2)\varepsilon_{<ae>} \circ (\mathrm{pop} \times 1_E) \circ (\mathrm{pop} \times 1_{Z \times E}) \circ [1243] \circ [1324] \circ (1_{S \times E} \times \mathrm{rel}_S) \quad \text{(P4e)}$$
$$= (\alpha_1)\varepsilon_{<ae>} \circ (\alpha_2)\varepsilon_{<ae>} \circ (\mathrm{pop} \times 1_E) \circ (\mathrm{pop} \times 1_{Z \times E}) \circ [1423] \circ (1_{S \times E} \times \mathrm{rel}_S) \quad \text{(P2)}$$
$$= (\alpha_1)\varepsilon_{<ae>} \circ (\alpha_2)\varepsilon_{<ae>} \circ (\mathrm{relI}) \quad \text{(F9)}$$
$$= ((\alpha_1)\varepsilon_{<ae>},(\alpha_2)\varepsilon_{<ae>})\mathrm{rel}_U \quad \text{(U15)}$$

(16) $((\beta)\neg_M)\varepsilon_{<be>} = (\beta \circ (\iota_2,\iota_1))\varepsilon_{<be>}$ (M16)
$$= 1_S \times (\beta \circ (\iota_2,\iota_1)) \quad \text{(B2)}$$
$$= (1_S \times \beta) \circ 1_S \times (\iota_2,\iota_1) \quad \text{(G2)}$$
$$= \beta\varepsilon_{<be>} \circ ((1_S \times \iota_2),(1_S \times \iota_1)) \quad \text{(B2,C2)}$$
$$= \beta\varepsilon_{<be>} \circ (x_2^2,x_1^2) \quad \text{(C1,S1)}$$
$$= (\beta\varepsilon_{<be>})\neg_U \quad \text{(U16)}$$

(17a) $((\beta_1,\beta_2)\wedge_M)\varepsilon_{<be>} = (\beta_1 \circ (\iota_1,\beta_2))\varepsilon_{<be>}$ (M17a)

$$= 1_S \times (\beta_1 \circ (\iota_1, \beta_2)) \quad \text{(B2)}$$

$$= (1_S \times \beta_1) \circ 1_S \times (\iota_1, \beta_2) \quad \text{(G2)}$$

$$= (\beta_1)\varepsilon_{<be>} \circ (1_S \times \iota_1, 1_S \times \beta_2) \quad \text{(B2,C2)}$$

$$= (\beta_1)\varepsilon_{<be>} \circ (1_S \times \iota_1, (\beta_2)\varepsilon_{<be>}) \quad \text{(B2)}$$

$$= (\beta_1)\varepsilon_{<be>} \circ (x_1^2, (\beta_2)\varepsilon_{<be>}) \quad \text{(S1,C1)}$$

$$= ((\beta_1)\varepsilon_{<be>}, (\beta_2)\varepsilon_{<be>})^{\wedge}{}_U \quad \text{(U17a)}$$

(17b) $((\beta_1,\beta_2)^{\vee}{}_M)\varepsilon_{<be>} = (\beta_1 \circ (\beta_2, \iota_2))\varepsilon_{<be>} \quad \text{(M17b)}$

$$= 1_S \times (\beta_1 \circ (\beta_2, \iota_2)) \quad \text{(B2)}$$

$$= (1_S \times \beta_1) \circ 1_S \times (\beta_2, \iota_2) \quad \text{(G2)}$$

$$= (\beta_1)\varepsilon_{<be>} \circ (1_S \times \beta_2, 1_S \times \iota_2) \quad \text{(B2,C2)}$$

$$= (\beta_1)\varepsilon_{<be>} \circ ((\beta_2)\varepsilon_{<be>}, 1_S \times \iota_2) \quad \text{(B2)}$$

$$= (\beta_1)\varepsilon_{<be>} \circ ((\beta_2)\varepsilon_{<be>}, x_2^2) \quad \text{(S1,C1)}$$

$$= ((\beta_1)\varepsilon_{<be>}, (\beta_2)\varepsilon_{<be>})^{\vee}{}_U \quad \text{(U17b)}$$

11. CONCLUSION

The eighteen proofs, yielding the homomorphism property of ε, turned out to be considerable longer and more cumbersome than we had expected. But they are equational and we believe that we have isolated the properties used for the correctness proof. That list of properties is itself somewhat of a motley assortment and we feel that it can and should be cleaned up. We hope, however, that the reader will recognize that something very different is going on in that the compiler correctness is being developed in a machine checkable equational framework despite those rough edges.

Perhaps it is typical of detailed and exhaustive correctness efforts, but the process of carrying out the 18 proofs with unflinching detail uncovered several errors in the preceding definitions. This was particularly true of the more difficult (more lengthy) proofs involving the more complex definitions: 9, 12, and, 15. These proofs pointed to errors in the source definition of binary arithmetic operation evaluation (M9), of the block construct (M12), and in the definition of "switchI" in terms of pop and push (F9).

Note also the important fact that the 18 proofs are independent; that is, each programming feature is analyzed independent of the others. So long as the language can be extended within the semantic definition of Section 4, that extension can be checked without consideration of the rest of the correctness proof.

We hope, in the future, to carry out such extensions; even to classify what extensions are possible. Also, if the extension requires new semantic domains for the denotational semantics of the language (the carriers of M) we hope that there will be a uniform way to carry over the proofs already done.

Finally, we hope to carry out the same kind of algebraic arguments with alternative semantic definitions; alternatives to ψ (compile) and alternatives to θ (source semantics). One would hope also to find translations of the flow chart language so that correctness of a composite translation would be obtained by "pasting" commuting squares together.

ACKNOWLEDGEMENTS

We have had a continuing interest in the "complier correctness problem." In the spring of 1974 that interest was active; Susanna Ginali did a thorough study of the McCarthy and Painter (1967) and Burstall and Landin (1969) papers while visiting IBM. There were several fruitful discussions with Joe Goguen and Susanna Ginali at that time. An important discussion with Calvin Elgot occurred during the summer of 1978 at which time we realized that T_0 should be the category of flow charts, rather than the quotient rational theory or the continuous algebraic theory of countable trees. We are deeply indebted to Susanna Ginali, Joe Goguen and Calvin Elgot for their help and encouragement in general and for their contributions to our progress on this problem in particular.

This work on compiler correctness was initiated following a series of lectures on algebraic semantics for the Summer School on Foundations of Artificial Intelligence and Computer Science, Pisa, Italy, 19-30 June 1978, by JWT. We were seeking a significant and informative example employing many of the algebraic concepts. This example (though not developed to the point it is here) was used for the 3rd Advanced Course on Foundations of Computer Science, Amsterdam, The Netherlands, 21 August - 1 September, 1978. An earlier version of this paper was presented at the Sixth International Colloquium on Automata, Languages and Programming in Graz, Austria, July, 1979. We are grateful to the organizers and sponsors of the summer schools and the colloquium for the opportunity to discuss and promote our ideas on algebraic semantics.

We are also very grateful to Andrzej Blikle, Steve Bloom, Calvin Elgot, Marie-Claude Gaudel, Robin Milner, Barry Rosen and the ICALP '79 and TCS referees for discussions, specific suggestions and corrections based on earlier drafts of this paper.

BIBLIOGRAPHY

ADJ (Authors: J. A. Goguen, J. W. Thatcher, E. G. Wagner and J. B. Wright)

(1975) (JAG, JWT, EGW, JBW) "Initial algebra semantics and continuous algebras," IBM Research Report RC-5701. November 1975. *JACM 24* (1977) pp. 68-95.

(1976) (JWT, EGW, JBW) "Specification of abstract data types using conditional axioms," IBM Research Report RC-6214, September 1976.

(1976a) (JAG, JWT, EGW) "An initial algebra approach to the specification, correctness, and implementation of abstract data types," IBM Research Report RC-6487, October 1976. To appear, *Current Trends in Programming Methodology, IV: Data Structuring* (R. Yeh, Ed.), pp. 80-149, Prentice Hall, New Jersey.

(1976b) (EGW, JBW, JAG, JWT) "Some fundamentals of order algebraic semantics," *Lecture Notes in Computer Science 45* (Mathematical Foundations of Computer Science 1976), Springer-Verlag, pp. 153-168; IBM Research Report RC 6020, May 1976.

(1976c) (JBW, JWT, EGW, JAG) "Rational algebraic theories and fixed-point solutions," *Proceedings* 17th IEEE Symposium on Foundations of Computing, Houston, Texas, October, 1976, pp. 147-158.

(1977) (EGW, JWT, JBW) "Free continuous theories," IBM Research Report RC 6906, December 1977. Accepted for publication, *Fundamenta Informaticae.*

Arbib, M.A. and Giveon, Y.

(1968) "Algebra automata I: Parallel programming as a prolegomena to the categorical approach," *Information and Control 12* (1968) 331-345.

Birkhoff, G. and Lipson, J.D.

(1970) "Heterogeneous algebras," *J. Combinatorial Theory 8* (1970) 115-133.

Burstall, R.M. and Landin, P.J.

(1969) "Programs and their proofs: an algebraic approach," *Machine Intelligence 4*, 1969.

Elgot, C.C.

(1973) "Monadic computation and iterative algebraic theories," IBM Research Report RC 4564, October 1973. *Proceedings*, Logic Colloquium 1973, North Holland (1975) 175-230.

(1977) "Some geometrical categories associated with flow chart schemes," IBM Research Report RC 6534, May 1977. *Proceedings*, Conference on Fundamentals of Computation Theory, Poznan-Kornik, Poland, 1977.

Elgot, C.C. and Shepherdson, J.C.

(1977) "A semantically meaningful characterization of reducible flow chart schemes," IBM Research Report RC 6656, July, 1977.

Fiebrich, Rolf-Dieter

(1978) "Generation of correct compiler parts from formal language descriptions," LRZ-Bericht Nr. 7802/1, Institut für Informatik der Ludwig- Maximilians-Universität, Müchen, 1978.

Gaudel, M.C.

(1980) "Specification of compilers as abstract data type representations," draft manuscript, IRIA, Paris France. Presented, Workshop on Semantics Directed Compiler Generation, Aarhus, Denmark, January 1980.

(1980a) Thesis, March, 1980.

Gremano, G. and Maggiolo-Schettini, A.

(1975) "Proving a compiler correct: A simple approach," *JCSS 10* (1975) 370-383.

Guttag, J. V.

(1975) "The specification and application to programming of abstract data types," Univ. of Toronto, Computer Systems Research Group, Technical Report CSRG-59, September, 1975.

Lawvere, F.W.

(1963) "Functorial semantics of algebraic theories," *Proceedings*, Nat'l Acad. Sci. *50* (1963) 869-872.

McCarthy, J. and Painter, J.

(1967) "Correctness of a compiler for arithmetic expressions," *Mathematical Aspects of Computer Science*, Proceedings of Symposia in Applied Mathematics, Vol. 19 (J.T. Schwartz, Ed.)

American Math. Soc., Providence R.I. (1967) 33-41.

Milner, R.

(1972) "Implementation and application of Scott's logic for computable functions," *Proceedings*, ACM Conference on Proving Assertions about Programs, Las Cruces, New Mexico, January, 1972, pp. 1 - 6.

(1976) "Program semantics and mechanized proof," Mathematical Centre Tracts 82 (K.R. Apt and J.W. de Bakker (Eds.), Mathematisch Centrum, Amsterdam, 1976, pp. 3-44.

Milner, R. and Weyrauch, R

(1972) "Proving compiler correctness in a mechanized logic," *Machine Intelligence 7* (B. Meltzer and D. Michie, Eds.), Edinburgh University Press (1972) 51-72.

Morris, F. L.

(1972) "Correctness of translations of programming languages," Stanford Computer Science Memo CS 72-303 (1972).

(1973) "Advice on structuring compilers and proving them correct," *Proceedings*, ACM Symposium on Principles of Programming Languages, Boston (1973) 144-152.

Mosses, P.

(1977) "Making denotational semantics less concrete," manuscript, Aarhus University, August, 1977.

(1978) "Modular denotational semantics," Draft paper, 1978-11-11, Department of Computer Science, Institute of Mathematics, Aarhus University, 1978.

(1979) "A constructive approach to compiler correctness," draft manuscript, Department of Computer Science, Aarhus University, November 1979. Presented, Workshop on Semantics Directed Compiler Generation, Aarhus, January, 1980.

Schmeck, Hartmut

(1975) "Korrektheit von Übersetzungen," Bericht Nr. 3/75 des Institut für Infromatik und Praktische Mathematik, Christian-Albrechts-Universität Kiel, 1975.

Scott, D. and Strachey, C.

(1971) "Toward a mathematical semantics for computer languages," Technical Monograph PRG-6, Oxford University Computing Laboratory, Programming Research Group, 1971.

Zilles, S. N.

(1974) "Algebraic specification of data types," Computation Structures Group Memo 119, MIT, Cambridge, Mass. (1974) 28-52.