Semantics of the Domain of Flow Diagrams

JOHN C. REYNOLDS

Syracuse University, Syracuse, New York

ABSTRACT. A domain of flow diagrams similar to that proposed by Scott, a domain of linear flow diagrams proposed by Goguen et al., a domain of decision table diagrams involving infinitary branching, and a domain of processes based on the ideas of Milner and Bekic are each provided with a direct semantics, closely related to partial-function semantics, and a continuation semantics similar to that developed by Morris and Wadsworth. It is shown that there is a variety of meaning-preserving continuous functions among these language-like domains, that every direct semantics possesses an "equivalent" continuation semantics, and that there is a particular continuation semantics which always gives distinct meanings to distinct processes. The proofs utilize the algebraic methods of Goguen et al., which are extended to continuous algebras with operations whose arguments can be indexed by infinite sets or even domains.

KEY WORDS AND PHRASES: flow diagrams, lattices, domains, decision tables, processes, direct semantics, partial-function semantics, continuations, equivalence, algebraic semantics

CR CATEGORIES: 5.24

Introduction

Scott [12] has shown that a simple language of flow diagrams, in which primitive instructions are combined by composition and conditional operators, can be embedded in a complete lattice containing infinite diagrams which include expansions of loops and recursions. Goguen, Thatcher, Wagner, and Wright [4] have shown that this lattice can be viewed as an initial algebra, so that algebraic methods [2] can be used to define and relate its semantics. They have also proposed a distinct algebra of "linear" flow diagrams with a more limited form of composition operator.

In this paper we consider both kinds of flow diagram, as well as a domain of *decision table diagrams* involving infinitary branching, and a domain of *processes* suggested by the ideas of Milner [7] and Bekic [1]. For each of these "languages," we define a direct semantics, similar to the partial-function semantics used in the theory of schemas, and a continuation semantics, similar to that developed by Morris [8] and Wadsworth [15]. We exhibit a variety of meaning-preserving functions among these languages. We also show that every direct semantics possesses an "equivalent" continuation semantics and that there is a particular continuation semantics which always gives distinct meanings to distinct processes.

Our proofs utilize and illustrate the algebraic methods of Goguen et al. To facilitate the treatment of decision table diagrams and processes, we show that these methods can be extended to continuous algebras with operations whose arguments can be indexed by infinite sets or even domains.

Domains and Predomains

Our starting point is the so-called "lattice-theoretic approach" to the theory of computation, originally developed by Scott [11, 13]. The heart of this approach is the assumption

This work was supported by the National Science Foundation under Grant GJ-41540.

Author's address: School of Computer and Information Science, Syracuse University, Syracuse, NY 13210.

that domains of data can be partially ordered by a relationship $\sqsubseteq$ of *approximation*, in terms of which one can formulate a completeness property satisfied by the domains, and a continuity property satisfied by meaningful functions between domains. In Scott's own work [11–13] the completeness property is the existence of least upper bounds for all subsets of a domain, so that a domain is a complete lattice. Other authors [3–6, 9], however, have worked with a variety of weaker completeness properties such as the existence of least upper bounds for directed sets.

After considerable experimentation, we have decided to work within the framework of one of the weaker completeness properties. The use of complete lattices would introduce so-called "overdefined" domain elements, which do not possess any obvious computational reality and which preclude the simple formulation of several intuitively reasonable relationships, particularly the close connection between direct semantics and conventional partial-function semantics.

A subset of a partially ordered set is said to be *directed* iff it contains an upper bound for each of its finite subsets. A *predomain* is a partially ordered set in which every directed subset X possesses a least upper bound, written $\bigsqcup X$. A *domain* is a predomain which contains a least element, written $\perp$. A function f from a predomain S to a domain D is *continuous* iff $f(\bigsqcup X) = \bigsqcup\{f(x) \mid x \in X\}$ for all directed subsets X of S. A function from a domain to a domain is *strict* iff it preserves $\perp$.

There is a variety of ways in which we could alter these basic definitions with only minor changes in the ensuing development. We might use chains or countable chains instead of directed sets, or we might impose various topological restraints on domains, such as lattice continuity, algebracity, or the existence of countable bases. But the above definitions seem to provide the simplest adequate framework for our results.

In the terminology of [4], our domains are strict Δ-complete posets, and our continuous functions are Δ-continuous. Perhaps the use of Scott's own term "domain" with a different definition is presumptuous, but we want to invoke the connotations of Scott's terminology.

The relatively unfamiliar concept of a predomain plays a central role in our development. An interesting portion of the lattice-theoretic approach extends to predomains, and they are useful intermediaries in the construction of domains. Most important, the concept includes both domains and ordinary sets, which we consider to be predomains partially ordered by their identity relations. The result is a significant unification. For example, in defining semantics we will use an unspecified predomain S of states. In conventional applications S will be an ordinary set, but the validity of Propositions 7 and 8 requires the use of an S which is a domain.

For a predomain S and a domain D, we write $S \to D$ to denote the set of continuous functions from S to D, partially ordered by $f \sqsubseteq g$ iff $(\forall x \in S)\ f(x) \sqsubseteq g(x)$. It can be shown that $S \to D$ is always a domain, in which $(\bigsqcup_{S\to D} F)(x) = \bigsqcup_D \{f(x) \mid f \in F\}$ for directed $f \subseteq S \to D$, and $\perp_{S\to D}(x) = \perp_D$. When S is a domain, $S \to D$ is the usual domain of continuous functions. At the other extreme, when S is a set (partially ordered by its identity relation), $S \to D$ is the domain of all functions from S to D.

For a predomain S, we write $S_\perp$ to denote the domain formed by adding a new least element to S, i.e. the disjoint union $\{\perp\} \cup S$ partially ordered by $x \sqsubseteq y$ iff $x = \perp$, $x \in S$ and $y \in S$ and $x \sqsubseteq_S y$. When S and S' are sets, there is an obvious isomorphism between $S \to S'_\perp$ and the domain of partial functions from S to S' partially ordered by the subset relation between the graphs of the partial functions.

We write Bool for the predomain $\{\textbf{true}, \textbf{false}\}$. Let S, S', S'' be predomains, D, D', D'' be domains, $e \in S \to S'_\perp$, $f \in S' \to S''_\perp$, $g \in S'' \to D$, $h \in D \to D'$, $i \in D' \to D''$, and $q \in S \to (S \to S'_\perp)$. Let ρ be a strict function in $D \to D'$. We define the following expressions:

$$I_D \equiv \lambda x.\, x \in D \to D,$$
$$h \cdot g \equiv \lambda s''.\, h(g(s'')) \in S'' \to D',$$
$$ext(g) \equiv \lambda x''.\ \textbf{if } x'' = \perp \textbf{ then } \perp \textbf{ else } g(x'') \in S''_\perp \to D,$$

$$J_S \equiv \lambda s.\, s \in S \to S_\perp,$$
$$g{*}f \equiv ext(g)\cdot f \in S' \to D,$$

$$cond_D \equiv \lambda p.\, \lambda x_1.\, \lambda x_2.\ \textbf{if}\ p = \begin{Bmatrix} \perp \\ \textbf{true} \\ \textbf{false} \end{Bmatrix}\ \textbf{then}\ \begin{Bmatrix} \perp \\ x_1 \\ x_2 \end{Bmatrix} \in \mathrm{Bool}_\perp \to (D \to (D \to D)).$$

Each of these expressions is continuous in all variables. The reader may verify that

$$\begin{aligned}
&h\cdot I_D = I_{D'}\cdot h = h,\\
&(i\cdot h)\cdot g = i\cdot(h\cdot g),\\
&ext(J_S) = I_{S_\perp},\\
&e{*}J_S = J_{S'}{*}e = e,\\
&(g{*}f){*}e = g{*}(f{*}e),\\
&\lambda s.\, (g{*}(q(s)))(s) = g{*}(\lambda s.\, q(s)(s)),\\
&cond_{S''\to D}(p, g_1, g_2)(s'') = cond_D(p, g_1(s''), g_2(s'')),\\
&\rho(cond_D(p, x_1, x_2)) = cond_{D'}(p, \rho(x_1), \rho(x_2)).
\end{aligned}$$

As illustrated by the last two equations, we often write $f(x_1, \ldots, x_n)$ for $f(x_1)\cdots(x_n)$.

Note that, under the isomorphism between $S \to S'_\perp$ and the domain of partial functions from a set S to a set S', the composition operator $*$ mirrors the usual composition of partial functions.

For predomains S_1 and S_2, we write $S_1 \times S_2$ to denote the predomain $\{\langle x_1, x_2\rangle \mid x_1 \in S_1$ and $x_2 \in S_2\}$ partially ordered by $\langle x_1, x_2\rangle \sqsubseteq \langle y_1, y_2\rangle$ iff $x_1 \sqsubseteq y_1$ and $x_2 \sqsubseteq y_2$. When both S_1 and S_2 are domains, $S_1 \times S_2$ is a domain with the least element $\langle\perp, \perp\rangle$.

For predomains S_1 and S_2, we write $S_1 + S_2$ to denote the domain $\{\perp\} \cup \{\langle 1, x_1\rangle \mid x_1 \in S_1\} \cup \{\langle 2, x_2\rangle \mid x_2 \in S_2\}$ partially ordered by $x \sqsubseteq y$ iff $x = \perp$, or $x = \langle i, x'\rangle$ and $y = \langle i, y'\rangle$ and $x' \sqsubseteq_{S_i} y'$. This usage is equivalent to defining $S_1 + S_2 = (S_1 \oplus S_2)_\perp$, where $\oplus$ denotes a conventional disjoint union of sets (with the obvious partial ordering).

We will need to generalize this kind of sum to an iterative construct. Suppose OP is a set and, for each $\sigma \in OP$, S_σ is a predomain. We write $\sum_{\sigma\in OP} S_\sigma$ to denote the domain $\{\perp\} \cup \{\langle\sigma, x'\rangle \mid \sigma \in OP$ and $x' \in S_\sigma\}$ partially ordered by $x \sqsubseteq y$ iff $x = \perp$, or $x = \langle\sigma, x'\rangle$ and $y = \langle\sigma, y'\rangle$ and $x' \sqsubseteq_{S_\sigma} y'$. When their operands are domains, $+$ and $\sum$ denote the usual notion of a separated sum of domains.

If $f_\sigma \in S_\sigma \to D'$ for each $\sigma \in OP$, we write $\sum^*_{\sigma\in OP} f_\sigma$ to denote the function $g \in (\sum_{\sigma\in OP} S_\sigma) \to D'$ such that $g(\perp) = \perp$ and $g(\langle\sigma, x'\rangle) = f_\sigma(x')$.

Algebras and Homomorphisms

We use a notion of algebra which is similar to the continuous algebras of Goguen, Thatcher, Wagner, and Wright [4]; however, we generalize this notion to permit operations whose arguments can be indexed by arbitrary, perhaps infinite, predomains. On the other hand, we do not explore the many-sorted case treated in [4] since the conventional one-sorted case is notationally simpler and adequate for our needs.

A *signature* Σ consists of a set OP of *operators* and a mapping *rank* which assigns a predomain to each operator in OP. For a predomain S, we write Σ_S to denote $\{\sigma \mid \sigma \in OP$ and $rank(\sigma) = S\}$. We will normally specify a signature by listing each nonempty Σ_S. A Σ-*algebra* ΣX consists of a domain X, called the *carrier* of ΣX, and an *interpretation* which assigns to each $\sigma \in \Sigma_S$ an operation $\Sigma X_\sigma \in (S \to X) \to X$.

In this formulation, conventional algebraic operations such as constant, unary, and binary operations are provided by the ranks $S = \{\ \}, \{1\}, \{1, 2\}, \ldots$. Strictly speaking, we should write

$$\left.\begin{matrix} \Sigma X_\sigma(\langle\ \rangle), \\ \Sigma X_\sigma(\langle x_1\rangle), \\ \Sigma X_\sigma(\langle x_1, x_2\rangle) \end{matrix}\right\}\ \text{instead of the conventional notation}\ \left\{\begin{matrix} \Sigma X_\sigma, \\ \Sigma X_\sigma(x_1), \\ \Sigma X_\sigma(x_1, x_2), \end{matrix}\right.$$

where $\langle x_1, \dots, x_n\rangle$ denotes the function from $\{1, \dots, n\}$ to X which maps i into x_i. However, we will frequently use the less cumbersome conventional notation. Since we regard $\Sigma X_\sigma(x_1, x_2)$ as an abbreviation for $\Sigma X_\sigma(x_1)(x_2)$, this usage is tantamount to identifying

$$\begin{array}{ll} (\{\ \} \to X) \to X & \text{with } X, \\ (\{1\} \to X) \to X & \text{with } X \to X, \\ (\{1, 2\} \to X) \to X & \text{with } X \to (X \to X). \end{array}$$

Algebras will usually be named by giving their signature and carrier. One must remember, however, that two algebras with the same signature and carrier can still have different interpretations. We use Σ for an arbitrary signature, and Ω, Λ, Γ, and Δ for specific signatures. Similarly, we use X (with occasional superscripts) for an arbitrary carrier and other symbols for specific carriers.

Let ΣX and $\Sigma X'$ be Σ-algebras and ρ be a strict continuous function from X to X'. If, for all $\sigma \in \Sigma_S$ and $\mathbf{x} \in S \to X$, ρ satisfies the *homomorphic equation* $\rho(\Sigma X_\sigma(\mathbf{x})) = \Sigma X'_\sigma(\rho \cdot \mathbf{x})$, then ρ is said to be a *homomorphism from* ΣX *to* $\Sigma X'$. We write $\Sigma X \to \Sigma X'$ for the set of such homomorphisms. When $S = \{1, \dots, n\}$, the identifications given above reduce the homomorphic equation to the conventional form $\rho(\Sigma X_\sigma(x_1, \dots, x_n)) = \Sigma X'_\sigma(\rho(x_1), \dots, \rho(x_n))$.

As usual, algebras and homomorphisms form a category, i.e.

$$\begin{array}{l} I_X \in \Sigma X \to \Sigma X, \\ \rho \in \Sigma X \to \Sigma X' \text{ and } \rho' \in \Sigma X' \to \Sigma X'' \text{ imply } \rho' \cdot \rho \in \Sigma X \to \Sigma X''. \end{array}$$

An algebra ΣX is said to be *initial* (*key*) if, for each algebra $\Sigma X'$ with the same signature, there is exactly one (at most one) homomorphism from ΣX to $\Sigma X'$. As is shown in [4], all initial algebras with the same signature are isomorphic; so we can speak of *the* initial algebra for Σ. We denote this algebra by $\Sigma\mathrm{In}\Sigma$, its carrier by $\mathrm{In}\Sigma$, and the unique homomorphism from it to $\Sigma X'$ by $\mu_{\Sigma X'}$.

Although these definitions of algebras and homomorphisms are unorthodox, one can still prove the following theorem, which is the sine qua non of algebraic semantics:

THEOREM 1. *There is an initial algebra $\Sigma In\Sigma$ for any signature Σ.*

PROOF. We first construct $\Sigma\mathrm{In}\Sigma$ and then show that it possess a unique homomorphism to any Σ-algebra. The carrier is obtained by using Scott's inverse limit construction to obtain a domain satisfying the isomorphism

$$\mathrm{In}\Sigma \approx \sum_{\sigma \in OP} (rank(\sigma) \to \mathrm{In}\Sigma), \tag{1}$$

where OP and $rank$ are the operator set and rank mapping of the signature Σ. The inverse limit construction is described in detail in [10] and, more abstractly, in [16]. Although these descriptions use the framework of complete lattices, the construction carries over without significant change to the present definitions.

In general an isomorphism such as (1) can have many solutions. The particular solution $\mathrm{In}\Sigma$ produced by the inverse limit construction (starting with the one-point domain as D_0) is uniquely characterized within an isomorphism by the following property [10]: The identity function $I_{\mathrm{In}\Sigma}$ is the least solution of the equation

$$I = \sum_{\sigma \in OP}^{*} (\lambda \mathbf{x}.\, \langle \sigma, I \cdot \mathbf{x}\rangle). \tag{2}$$

Here the parenthesized expression denotes a function from $rank(\sigma) \to \mathrm{In}\Sigma$ to $\Sigma_{\sigma \in OP}\, (rank(\sigma) \to \mathrm{In}\Sigma)$. Strictly speaking, (2) should be written as

$$I = \Phi^{-1} \cdot \Big(\sum_{\sigma \in OP}^{*} (\lambda \mathbf{x}.\, \langle \sigma, I \cdot \mathbf{x}\rangle)\Big) \cdot \Phi,$$

where Φ is the isomorphic function from the left side of (1) to the right side, but we adopt the practice of eliding Φ and its inverse.

To make $\mathrm{In}\Sigma$ into a Σ-algebra, we provide the following interpretation of the operators: For all $\sigma \in \Sigma_S$ and $\mathbf{x} \in S \to \mathrm{In}\Sigma$, $\Sigma\mathrm{In}\Sigma_\sigma(\mathbf{x}) = \langle\sigma, \mathbf{x}\rangle$.

Now suppose ΣX is any Σ-algebra. Let $I_0, I_1, \ldots \in \mathrm{In}\Sigma \to \mathrm{In}\Sigma$ and $\mu_0, \mu_1, \ldots \in \mathrm{In}\Sigma \to X$ be the functions such that

$$I_0 = \perp,$$
$$I_{n+1} = \sum_{\sigma \in OP}^{*} (\lambda\mathbf{x}.\ \langle\sigma, I_n \cdot \mathbf{x}\rangle),$$
$$\mu_0 = \perp,$$
$$\mu_{n+1} = \sum_{\sigma \in OP}^{*} (\lambda\mathbf{x}.\ \Sigma X_\sigma(\mu_n \cdot \mathbf{x})).$$

By Scott's least fixed point theorem, the I_n and μ_n are directed sequences such that $\bigsqcup_{n=0}^{\infty} I_n$ is the least solution of eq. (2) and is therefore the identity function for $\mathrm{In}\Sigma$, while $\bigsqcup_{n=0}^{\infty} \mu_n$ is the least solution of

$$\mu = \sum_{\sigma \in OP}^{*} (\lambda\mathbf{x}.\ \Sigma X_\sigma(\mu \cdot \mathbf{x})).$$

This equation shows that μ satisfies $\mu(\Sigma\mathrm{In}\Sigma_\sigma(\mathbf{x})) = \mu(\langle\sigma, \mathbf{x}\rangle) = \Sigma X_\sigma(\mu \cdot \mathbf{x})$ for each operator σ, while the definition of $\sum^*$ ensures that μ is strict. Thus $\mu \in \Sigma\mathrm{In}\Sigma \to \Sigma X$.

On the other hand, suppose $\rho \in \Sigma\mathrm{In}\Sigma \to \Sigma X$. We show that $\rho \cdot I_n = \mu_n$ by induction on n. For $n = 0$ we have $\rho \cdot \perp = \perp$ since $\rho(\perp) = \perp$. The induction step is

$$\begin{aligned}\rho \cdot I_{n+1} &= \sum_{\sigma \in OP}^{*} (\lambda\mathbf{x}.\ \rho(\langle\sigma, I_n \cdot \mathbf{x}\rangle)) = \sum_{\sigma \in OP}^{*} (\lambda\mathbf{x}.\ \rho(\Sigma\mathrm{In}\Sigma_\sigma(I_n \cdot \mathbf{x})))\\ &= \sum_{\sigma \in OP}^{*} (\lambda\mathbf{x}.\ \Sigma X_\sigma(\rho \cdot I_n \cdot \mathbf{x})) = \sum_{\sigma \in OP}^{*} (\lambda\mathbf{x}.\ \Sigma X_\sigma(\mu_n \cdot \mathbf{x})) = \mu_{n+1},\end{aligned}$$

where the first equality uses the strictness of ρ. But then the continuity of composition gives $\rho = \rho \cdot I_{\mathrm{In}\Sigma} = \rho \cdot (\bigsqcup_{n=0}^{\infty} I_n) = \bigsqcup_{n=0}^{\infty} \rho \cdot I_n = \bigsqcup_{n=0}^{\infty} \mu_n = \mu$. □

When every operator has a rank of the form $\{1, \ldots, n\}$, Theorem 1 coincides with [4, Cor. 4.10], and despite its very different construction, $\Sigma\mathrm{In}\Sigma$ is isomorphic to the initial algebra CT_Σ of [4].

The idea behind algebraic semantics is to regard a language as an initial algebra and its semantic function, i.e. the function which maps each element of the language into its meaning, as the homomorphism into some target algebra with the same signature. Indeed, since this homomorphism is unique, the semantic function is fixed by the specification of the target algebra itself.

In the framework of continuous algebras, however, an initial algebra is far richer than a conventional word algebra—our "languages" contain partially defined and (most mysteriously) infinite elements. But the imposition of strictness and continuity upon homomorphisms forces these elements to behave themselves, while their presence provides a profound capability explored by Scott: Concepts such as iteration and recursion can be viewed as purely syntactic mechanisms which permit finite language elements to abbreviate infinite ones.

Direct Semantics of General Flow Diagrams

We now embark on a tour of several closely related languages and semantics. The initial view is informal; we provide a reasonable concrete syntax for the initial algebras and describe semantics in the style of Scott and Strachey [14], which, as illustrated in [4, Sec. 3.2], is equivalent to substituting target algebra operations into the homomorphic equations.

Let F be some set of *primitive instructions* and B be some set of *Boolean expressions*. Then the language of *general flow diagrams* is the initial algebra $\Omega\mathrm{In}\Omega$ for the signature Ω such that $\Omega_{\{\ \}} = \{I\} \cup F$, $\Omega_{\{1,2\}} = \{;\} \cup B$. Using the concrete syntax provided by Scott, we write:

I	for	$\Omega \mathrm{In}\Omega_I$,
f	for	$\Omega \mathrm{In}\Omega_f$,
$x_1; x_2$	for	$\Omega \mathrm{In}\Omega_;(x_1, x_2)$,
$b \to x_1, x_2$	for	$\Omega \mathrm{In}\Omega_b(x_1, x_2)$.

This initial algebra is isomorphic to $CT_{\Sigma'}$ in [4, Sec. 5.2, pt. II]. Except for the omission of overdefined elements, it is similar to Scott's lattice of flow diagrams [12]. (Albeit with other minor differences: Our formulation causes us to distinguish $\perp; \perp$ from $\perp$ and to disallow elements of the form $\perp \to x_1, x_2$.)

Let S be some predomain of *states*. Then the *direct semantics* of general flow diagrams is provided by the semantic function $\mu_{\Omega H} \in \mathrm{In}\Omega \to H$ into the "semantic domain" $H = S \to S_\perp$ such that:

$$\begin{aligned}
&\mu_{\Omega H}(I) = J_S,\\
&\mu_{\Omega H}(f) = \mathscr{F}(f),\\
&\mu_{\Omega H}(x_1; x_2) = \mu_{\Omega H}(x_2) * \mu_{\Omega H}(x_1),\\
&\mu_{\Omega H}(b \to x_1, x_2) = \lambda s.\, cond_{S_\perp}(\mathscr{B}(b, s), \mu_{\Omega H}(x_1, s), \mu_{\Omega H}(x_2, s)).
\end{aligned}$$

Here $\mathscr{F} \in F \to H$ and $\mathscr{B} \in B \to (S \to \mathrm{Bool}_\perp)$ are unspecified functions which provide the meaning of primitive instructions and Boolean expressions. Informally, these equations can be regarded as a language definition in the style of Scott and Strachey [14]. But from the algebraic viewpoint, they are simply the homomorphic equations which assert that $\mu_{\Omega H}$ is the unique homomorphism from $\Omega\mathrm{In}\Omega$ into a certain target algebra ΩH with carrier H. The structure of target algebras such as ΩH will be given more abstractly later.

When S is a set, $H = S \to S_\perp$ is isomorphic to the set of partial functions from S to S, and $*$ mirrors the usual composition of partial functions. In this case direct semantics reduces to the usual kind of partial-function semantics encountered in the treatment of schemas.

On the other hand, our direct semantics is intentionally more restrictive than the semantics suggested by Scott [12], in which the semantic domain is $S \to S$ for an unspecified complete lattice S, and $*$ is replaced by conventional functional composition. Scott's semantics becomes unnatural when one does not require $\mathscr{F}(f)$ to be strict. For example, let δ be a flow diagram whose meaning is $\perp$, presumably a diagram whose execution never terminates. Then $\delta; f$ could have a meaning different from $\perp$, which would suggest that the statement following a nonterminating statement could affect the computation. The obvious solution is to replace $S \to S$ and $S \to \mathrm{Bool}_\perp$ by domains of strict functions. Our direct semantics is basically similar to imposing this strictness requirement, but it emphasizes that $\perp$ in $S_\perp$ is not really a "state."

Direct semantics is capable of describing schema-like languages involving assignment and side-effect-free expressions. But is is inadequate for a variety of primitive instructions occurring in real programming languages. Let δ be a flow diagram whose meaning is $\perp$ (in the domain $S \to S_\perp$). Then for any primitive instruction f, the meaning $\mu_{\Omega H}(f; \delta)$ of $f; \delta$ will also be $\perp$. (Note that in contrast to the previous paragraph, we are now examining the effect of a primitive instruction which *precedes* a nonterminating diagram and can sensibly affect the computation.) This is clearly inadequate to accommodate primitive instructions such as **stop** or **print**(n). The introduction of such instructions precludes the assumption, made in direct semantics, that the meaning of an instruction is a function which accepts the state existing immediately prior to execution of the instruction and produces the state existing immediately after execution of the instruction. To avoid this assumption we turn to continuation semantics.

Continuation Semantics of General Flow Diagrams

In continuation semantics, originally developed by Morris [8] and Wadsworth [15], the meaning of an instruction (or flow diagram) is a function which accepts the state existing immediately prior to execution, plus an additional argument called the *continuation*, and produces the final output of the entire program. The continuation which is provided as an additional argument is a function from the state existing after instruction execution to the final program output, which gives the semantics of the "rest of the computation" to be performed if the current instruction "terminates normally." Thus an instruction with normal behavior will produce its output by applying the continuation to the state following execution. But an "abnormal" instruction can produce the final output in some other manner – possibly ignoring the continuation.

Let S again be some predomain of states, and let O be some domain of *outputs*, whose least element $\perp$ denotes the "output" of a nonterminating computation. The domain of *continuations* is $C = S \rightarrow O$, and the semantic domain is $W = C \rightarrow (S \rightarrow O) = C \rightarrow C$. (Somewhat counterintuitively, we have made continuations the first argument and states the second argument of the meanings of flow diagrams; this arrangement will eventually simplify our semantic equations.) Then the *continuation semantics* of general flow diagrams is provided by the semantic function $\mu_{\Omega W} \in \text{In}\Omega \rightarrow W$ such that:

$$\begin{aligned}
&\mu_{\Omega W}(I) = \lambda c.\ \lambda s.\ c(s),\\
&\mu_{\Omega W}(f) = \mathscr{G}(f),\\
&\mu_{\Omega W}(x_1; x_2) = \lambda c.\ \lambda s.\ \mu_{\Omega W}(x_1, \lambda s'.\ \mu_{\Omega W}(x_2, c, s'), s),\\
&\mu_{\Omega W}(b \rightarrow x_1, x_2) = \lambda c.\ \lambda s.\ cond_O(\mathscr{B}(b, s),\ \mu_{\Omega W}(x_1, c, s),\ \mu_{\Omega W}(x_2, c, s)),
\end{aligned}$$

where $\mathscr{G} \in F \rightarrow W$ and $\mathscr{B} \in B \rightarrow (S \rightarrow \text{Bool}_\perp)$ are unspecified functions providing the semantics of primitive instructions and Boolean expressions.

The essence of continuation semantics is revealed by the third equation. Intuitively, to execute $x_1; x_2$ with an initial state s and a continuation c, we execute x_1 with the state s and a continuation $\lambda s'.\ \mu_{\Omega W}(x_2, c, s')$, which picks up the state s' after execution of x_1 and then executes x_2 with s' and the continuation c, which in turn picks up the state after execution of x_2 and then executes the rest of the program. But more generally, the equation shows that it is the meaning $\mu_{\Omega W}(x_1)$ of x_1 which determines how the final output will be affected by the meaning $\mu_{\Omega W}(x_2)$ of x_2, which in turn determines how the final output will be affected by the "meaning of the rest of the program" c.

Further insight is provided by a brief digression on the semantics of some "abnormal" primitive instructions. To handle **stop** $\in F$, we can take $O = S_\perp$ and $\mathscr{G}(\textbf{stop}) = \lambda c.\ \lambda s.\ s = \lambda c.\ J_S$. This choice makes it clear that the final output caused by a stop instruction will be the state existing immediately before its execution, regardless of the rest of the program.

To handle intermediate output of integers, let Int be the set of integers, N be some set of integer expressions, and $\mathscr{N} \in N \rightarrow (S \rightarrow \text{Int}_\perp)$ be a function giving the semantics of integer expressions. Let O be a domain satisfying the isomorphism $O \approx S + \text{Int} \times O$ and let

$$\begin{aligned}
&\mathscr{G}(\textbf{print}\, n) = \lambda c.\ \lambda s.\ ext(\lambda i \in \text{Int}.\ \rho_2(i, c(s)))(\mathscr{N}(n, s)),\\
&\mathscr{G}(\textbf{stop}) = \lambda c.\ \lambda s.\ \rho_1(s),
\end{aligned}$$

where ρ_1 and ρ_2 are the obvious injection functions from S and $\text{Int} \times O$, respectively, into O. The elements of O can be classified into three distinct kinds of output:

(1) $\rho_2(i_1, \ldots, \rho_2(i_k, \rho_1(s)) \ldots)$, which would be the output of a program which prints the integers $i_1, \ldots, i_k$ and then terminates in the state s;

(2) $\rho_2(i_1, \ldots, \rho_2(i_k, \perp) \ldots)$, which would be the output of a program which prints the integers $i_1, \ldots, i_k$ and then runs forever without further printing;

(3) $\rho_2(i_1, \rho_2(i_2, \ldots))$ – a limit point in the domain O – which would be the output of a program which prints the endless sequence of integers $i_1, i_2, \ldots$.

In this situation, "final" output is a misnomer since a program can continue to generate output forever. A better term would be *irreversible* output since the semantics ensures that the rest of the program cannot rescind the effect of a print instruction.

Returning to general flow diagrams, where the interpretation of primitive instructions is left unspecified, we can simplify our equations for $\mu_{\Omega W}$ by using eta-reduction.

Specifically, $\lambda s.\, c(s) = c$, $\lambda s'.\, \mu_{\Omega W}(x_2, c, s') = \mu_{\Omega W}(x_2, c)$, and $\lambda s.\, \mu_{\Omega W}(\ldots, s) = \mu_{\Omega W}(\ldots)$. Thus

$$\begin{aligned}
&\mu_{\Omega W}(I) = \lambda c.\, c = I_C,\\
&\mu_{\Omega W}(f) = \mathcal{G}(f),\\
&\mu_{\Omega W}(x_1; x_2) = \lambda c.\, \mu_{\Omega W}(x_1, \mu_{\Omega W}(x_2, c)) = \mu_{\Omega W}(x_1)\cdot \mu_{\Omega W}(x_2),\\
&\mu_{\Omega W}(b \to x_1, x_2) = \lambda c.\, \lambda s.\, cond_O(\mathcal{B}(b, s),\ \mu_{\Omega W}(x_1, c, s),\ \mu_{\Omega W}(x_2, c, s)).
\end{aligned}$$

Intriguingly, the order of composition in the third equation is the reverse of the order for direct semantics.

Linear Flow Diagrams

There are many flow diagrams which possess the same meaning, regardless of the choice of direct or continuation semantics. The other languages we consider can be thought of as successive attempts to strip away this redundancy and approach the goal of canonicality, where distinct diagrams have distinct meanings. Eventually, it will become clear that we have a succession of languages with meaning-preserving mappings from each language to the next, where the final language (of processes) is canonical for continuation semantics. For the present, however, this image is only meant to motivate our definitions, and no attempt will be made to prove relationships between languages or equivalences within a language.

In any reasonable semantics, one would expect the meaning of general flow diagrams to satisfy the following equivalences:

$$\begin{aligned}
&I; x_1 \simeq x_1,\\
&(x_1; x_2); x_3 \simeq x_1; (x_2; x_3),\\
&(b \to x_1, x_2); x_3 \simeq b \to (x_1; x_3), (x_2; x_3).
\end{aligned}$$

Intuitively—neglecting any complications which might be caused by infinite diagrams—these equivalences can be used to transform any flow diagram until every left operand of ";" is a primitive instruction. A flow diagram which meets this restriction is said to be *linear*.

Following Goguen, Thatcher, Wagner, and Wright [4], we can formulate linear flow diagrams as an initial algebra by regarding $f; x$ as the application of a unary operator, named by f, to the operand x. Thus the language of linear flow diagrams is the initial algebra $\Lambda \mathrm{In}\Lambda$ for the signature Λ such that $\Lambda_{\langle\,\rangle} = \{I\}$, $\Lambda_{\langle 1\rangle} = F$, $\Lambda_{\langle 1,2\rangle} = B$. This initial algebra is isomorphic to CT_Σ in [4, Sec. 5.2, pt. I]. As a concrete syntax, we write

I	for	$\Lambda\mathrm{In}\Lambda_I$,
$f; x$	for	$\Lambda\mathrm{In}\Lambda_f(x)$,
$b \to x_1, x_2$	for	$\Lambda\mathrm{In}\Lambda_b(x_1, x_2)$.

This notation has been chosen so that general and linear flow diagrams with the same concrete representation should have the same meaning. Intuitively, applying this relationship to the forms I, $f; x$, and $b \to x_1, x_2$ determines the semantic equations for linear flow diagrams.

Thus direct semantics is provided by the semantic function $\mu_{\Lambda H} \in \mathrm{In}\Lambda \to H$ such that

$$\begin{aligned}
&\mu_{\Lambda H}(I) = J_S,\\
&\mu_{\Lambda H}(f; x) = \mu_{\Lambda H}(x) * \mathcal{F}(f),\\
&\mu_{\Lambda H}(b \to x_1, x_2) = \lambda s.\, cond_{S_\perp}(\mathcal{B}(b, s),\ \mu_{\Lambda H}(x_1, s),\ \mu_{\Lambda H}(x_2, s)),
\end{aligned}$$

while continuation semantics is provided by $\mu_{\Lambda W} \in \mathrm{In}\Lambda \to W$ such that

$$\begin{aligned}
&\mu_{\Lambda W}(I) = I_C,\\
&\mu_{\Lambda W}(f; x) = \mathcal{G}(f)\cdot \mu_{\Lambda W}(x),\\
&\mu_{\Lambda W}(b \to x_1, x_2) = \lambda c.\, \lambda s.\, cond_O(\mathcal{B}(b, s),\ \mu_{\Lambda W}(x_1, c, s),\ \mu_{\Lambda W}(x_2, c, s)).
\end{aligned}$$

Decision Table Diagrams

There is a variety of equivalences for conditional branching operations. For example,

$$(b_1 \to (b_2 \to x_{11}, x_{12}), (b_2 \to x_{21}, x_{22})) \simeq (b_2 \to (b_1 \to x_{11}, x_{21}), (b_1 \to x_{12}, x_{22}))$$

or

$$(b_1 \to (b_2 \to x_{11}, x_{12}), (b_1 \to x_{21}, x_{22})) \simeq (b_1 \to (b_2 \to x_{11}, x_{12}), x_{22}).$$

As a step toward eliminating this kind of redundancy, we replace the set B of binary branching operations by a single many-way branching operation D, which is an idealization of the well-known programming concept of a *decision table*.

Suppose for a moment that $B = \{b_1, b_2\}$ has only two members. Then either side of the first equivalence given above can be replaced by

$$D\left[\begin{array}{c|ccc} {}_{b_1}\diagdown^{b_2} & \textbf{true} & \textbf{false} & \perp \\ \hline \textbf{true} & x_{11} & x_{12} & \perp \\ \textbf{false} & x_{21} & x_{22} & \perp \\ \perp & \perp & \perp & \perp \end{array}\right].$$

Similarly, either side of the second equivalence can be replaced by

$$D\left[\begin{array}{c|ccc} {}_{b_1}\diagdown^{b_2} & \textbf{true} & \textbf{false} & \perp \\ \hline \textbf{true} & x_{11} & x_{12} & \perp \\ \textbf{false} & x_{22} & x_{22} & x_{22} \\ \perp & \perp & \perp & \perp \end{array}\right].$$

Essentially, the decision $D[\mathbf{x}]$ means "Produce a list t of the current value of all Boolean expressions in B, and then execute the table entry $\mathbf{x}(t)$."

The "list" t is really a function in the domain $T = B \to \mathrm{Bool}_\perp$. Thus, if table entries belong to the set $\mathrm{In}\Gamma$ (which is going to be the language of decision table diagrams), then the decision table $\mathbf{x}$ itself will be a function from T to $\mathrm{In}\Gamma$. But only some of these functions are reasonable.

In the evaluation of any compound conditional branching operation, for a particular value of b_1, either b_2 is evaluated, so that the nontermination of b_2 implies the nontermination of the entire branching operation, or b_2 is not evaluated, so that the outcome is independent of b_2. This constraint is reflected by the fact that every row in a decision table has either the form $x\ y\ \perp$ or the form $x\ x\ x$. A similar rule holds for columns. As a consequence, decision tables are always monotonic functions from T to $\mathrm{In}\Gamma$.

The generalization to arbitrary finite B is straightforward. More surprisingly, we can even permit B to be infinite. The only qualification, which is typical of the lattice-theoretic approach, is that decision tables are required to be continuous, rather than merely monotonic, functions from T to $\mathrm{In}\Gamma$. Thus the decision operator D accepts operands which belong to $T \to \mathrm{In}\Gamma$, i.e. it is an operator with rank T. Although such an operator goes beyond the framework of conventional algebra, it is encompassed by our generalization to operators whose ranks are arbitrary predomains.

Of course an infinite decision table cannot be explicitly tabulated, but it can still be described by functional notation. For example, the decision table diagrams given above, when generalized to an arbitrary B containing b_1 and b_2, can be represented by

$$D[\lambda t.\ cond_{\mathrm{In}\Gamma}(t(b_1), cond_{\mathrm{In}\Gamma}(t(b_2), x_{11}, x_{12}), cond_{\mathrm{In}\Gamma}(t(b_2), x_{21}, x_{22}))]$$

and

$$D[\lambda t.\ cond_{\mathrm{In}\Gamma}(t(b_1), cond_{\mathrm{In}\Gamma}(t(b_2), x_{11}, x_{12}), x_{22})].$$

However, such expressions are not in themselves decision table diagrams, but only indirect and nonunique representations of such diagrams.

With this motivation, we can give a precise definition. The language of *decision table* *diagrams* is the initial algebra $\Gamma\text{In}\Gamma$ for the signature Γ such that $\Gamma_{\{\}} = \{I\}$, $\Gamma_{\{1\}} = F$, $\Gamma_T = \{D\}$. As a concrete syntax, we write

$$\begin{array}{lll} I & \text{for} & \Gamma\text{In}\Gamma_I, \\ f;x & \text{for} & \Gamma\text{In}\Gamma_f(x), \\ D[\mathbf{x}] & \text{for} & \Gamma\text{In}\Gamma_D(\mathbf{x}). \end{array}$$

Direct semantics is provided by the semantic function $\mu_{\Gamma H} \in \text{In}\Gamma \to H$ such that

$$\begin{aligned} &\mu_{\Gamma H}(I) = J_S, \\ &\mu_{\Gamma H}(f;x) = \mu_{\Gamma H}(x) * \mathcal{F}(f), \\ &\mu_{\Gamma H}(D[\mathbf{x}]) = \lambda s.\ \mu_{\Gamma H}(\mathbf{x}(\lambda b.\ \mathcal{B}(b,s)), s). \end{aligned}$$

In the last equation, the function $\lambda b.\ \mathcal{B}(b, s) \in T$ denotes the "list" of the values of all Boolean expressions in the state s.

For continuation semantics, we have the semantic function $\mu_{\Gamma W} \in \text{In}\Gamma \to W$ such that

$$\begin{aligned} &\mu_{\Gamma W}(I) = I_C, \\ &\mu_{\Gamma W}(f;x) = \mathcal{G}(f) \cdot \mu_{\Gamma W}(x), \\ &\mu_{\Gamma W}(D[\mathbf{x}]) = \lambda c.\ \lambda s.\ \mu_{\Gamma W}(\mathbf{x}(\lambda b.\ \mathcal{B}(b,s)), c, s). \end{aligned}$$

Admittedly we are stretching a point in calling decision table diagrams a language. They are even further than flow diagrams from the conventional finitary concept of language. But they still have the essential linguistic characteristic of being uninterpreted: They make no commitment to a choice of the set of states, nor to the meaning of primitive instructions or Boolean expressions, nor even to the choice between direct and continuation semantics.

Processes

There are two important equivalences for decision table diagrams. The first, $D[\lambda t.\ x] \simeq x$, shows that a decision is redundant if all of its table entries are the same. The second,

$$D[\lambda t.\ D[\lambda t'.\ g(t, t')]] \simeq D[\lambda t.\ g(t, t)],$$

where $g \in T \to (T \to \text{In}\Gamma)$, shows that, when the entries of a decision table are themselves decisions, the inner decisions must "go the same way" as the outer one, since there is no intervening primitive instruction which might change the state of the computation.

To eliminate this kind of redundancy, we define a further language in which decisions and primitive instructions are required to alternate. More precisely, a decision table entry will always have the form I, or $\perp$, or f; $D[\mathbf{x}]$ where $\mathbf{x}$ is a decision table.

To obtain an algebraic formulation, we regard f; $D[\mathbf{x}]$ as the application of a T-ary operator, named by f, to the operand $\mathbf{x}$. Then table entries are the initial algebra $\Delta\text{In}\Delta$ for the signature Δ such that $\Delta_{\{\}} = \{I\}$, $\Delta_T = F$. As a concrete syntax, we write

$$\begin{array}{lll} I & \text{for} & \Delta\text{In}\Delta_I, \\ f;D[\mathbf{x}] & \text{for} & \Delta\text{In}\Delta_f(\mathbf{x}), \end{array}$$

which suggests the relationship between our new language and the language of decision table diagrams.

If $\text{In}\Delta$ is the domain of table entries, then decision tables themselves belong to the domain $T \to \text{In}\Delta$, which we call Z. From eq. (1) in the proof of Theorem 1, $\text{In}\Delta$ satisfies the domain isomorphism

$$\begin{aligned} \text{In}\Delta &\approx \sum_{\sigma \in \{I\} \cup F} (\mathit{rank}(\sigma) \to \text{In}\Delta) \\ &= \sum_{\sigma \in \{I\} \cup F} \textbf{if } \sigma = I \textbf{ then } (\{\ \} \to \text{In}\Delta) \textbf{ else } (T \to \text{In}\Delta) \\ &= \sum_{\sigma \in \{I\} \cup F} \textbf{if } \sigma = I \textbf{ then } \{\cdot\} \textbf{ else } Z, \end{aligned}$$

where $\{\cdot\}$ denotes a one-element domain. But the right side is easily seen to be isomorphic to $\{\cdot\} + F \times Z$. Thus Z and $\text{In}\Delta$ satisfy

$$Z = T \to \text{In}\Delta, \quad \text{In}\Delta \approx \{\cdot\} + F \times Z.$$

These "domain equations" suggest a close connection with the concept of *processes* developed by Milner [7] and Bekic [1], which in fact inspired the language described here. To emphasize this connection we henceforth call Z the domain of *processes* and $\text{In}\Delta$ the domain of *process components*. It should be noted, however, that the processes of Milner and Bekic are less syntactic than ours and are specifically oriented to problems of concurrent processing which are not considered here.

Intuitively the process z means "For the current state s, compute a list $t = \lambda b.\, \mathscr{B}(b, s)$ of values of the Boolean expressions, and then execute the process component $z(t)$." The process component I means "Do nothing," while the process component $f;\, D[z]$ means "Execute f and then execute the process z."

This intuition is captured in direct semantics by the semantic functions $\delta_{\Gamma H} \in Z \to H$ and $\mu_{\Delta H} \in \text{In}\Delta \to H$ such that

$$\begin{aligned}
&\delta_{\Gamma H}(z) = \lambda s.\, \mu_{\Delta H}(z(\lambda b.\, \mathscr{B}(b, s)), s),\\
&\mu_{\Delta H}(I) = J_S,\\
&\mu_{\Delta H}(f;\, D[z]) = \delta_{\Gamma H}(z) * \mathscr{F}(f).
\end{aligned}$$

For continuation semantics, the semantic functions are $\delta_{\Gamma W} \in Z \to W$ and $\mu_{\Delta W} \in \text{In}\Delta \to W$ such that

$$\begin{aligned}
&\delta_{\Gamma W}(z) = \lambda c.\, \lambda s.\, \mu_{\Delta W}(z(\lambda b.\, \mathscr{B}(b, s)), c, s),\\
&\mu_{\Delta W}(I) = I_C,\\
&\mu_{\Delta W}(f;\, D[z]) = \mathscr{G}(f) \cdot \delta_{\Gamma W}(z).
\end{aligned}$$

For either set of semantic equations, the substitution of the first equation into the third gives a pair of homomorphic equations for $\mu_{\Delta H}$ or $\mu_{\Delta W}$. We have introduced the subsidiary functions $\delta_{\Gamma H}$ and $\delta_{\Gamma W}$ (whose names will become meaningful later) to treat the domains Z and $\text{In}\Delta$ more symmetrically.

Nevertheless, Z, unlike $\text{In}\Delta$, is not the carrier of an initial algebra. This anomaly is the price of avoiding many-sorted algebras. One could define Z and $\text{In}\Delta$ as the carriers of a two-sorted initial algebra, with $\delta_{\Gamma H}$ and $\mu_{\Delta H}$ (or $\delta_{\Gamma W}$ and $\mu_{\Delta W}$) as the components of a two-sorted homomorphism.

Algebraic Treatment of Decision Table Diagrams

From an algebraic viewpoint, the eight language definitions we have given are tantamount to a specification of the following target algebras:

Language	Direct semantics	Continuation semantics
General flow diagrams	ΩH	ΩW
Linear flow diagrams	ΛH	ΛW
Decision table diagrams	ΓH	ΓW
Process components	ΔH	ΔW

Our main task is to define these target algebras more directly and to investigate relationships among them. As each target algebra is defined, it will become apparent that the corresponding language definition is a display of the equations for the homomorphism into the target algebra from the initial algebra with the same signature.

We begin with decision table diagrams, then move "forward" to processes, and then move "backward" to linear and finally general flow diagrams.

Decision table diagrams are the initial algebra with the signature Γ such that $\Gamma_{()} = \{I\}$, $\Gamma_{(1)} = F$, and $\Gamma_T = \{D\}$, where $T = B \to \text{Bool}_\perp$. To describe their direct semantics, let S be some predomain of states, $H = S \to S_\perp$, $\mathscr{F} \in F \to H$, and $\mathscr{B} \in \text{B} \to (\text{S} \to \text{Bool}_\perp)$.

Then the target algebra ΓH is the Γ-algebra with carrier H such that

$$\begin{aligned}
&\Gamma H_I = J_S,\\
&\Gamma H_f(h) = h * \mathscr{F}(f),\\
&\Gamma H_D(\mathbf{h}) = \lambda s.\ \mathbf{h}(\lambda b.\ \mathscr{B}(b, s), s).
\end{aligned}$$

For continuation semantics, let S be some predomain of states, O be some domain of outputs, $C = S \to O$, $W = C \to C$, $\mathscr{G} \in F \to W$, and $\mathscr{B} \in B \to (S \to \mathrm{Bool}_\perp)$. Then the target algebra ΓW is the Γ-algebra with carrier W such that

$$\begin{aligned}
&\Gamma W_I = I_C,\\
&\Gamma W_f(w) = \mathscr{G}(f)\cdot w,\\
&\Gamma W_D(\mathbf{w}) = \lambda c.\ \lambda s.\ \mathbf{w}(\lambda b.\ \mathscr{B}(b, s), c, s).
\end{aligned}$$

The previously given semantic equations for decision table diagrams are simply the homomorphic equations for the unique homomorphisms $\mu_{\Gamma H} \in \Gamma\mathrm{In}\Gamma \to \Gamma H$ and $\mu_{\Gamma W} \in \Gamma\mathrm{In}\Gamma \to \Gamma W$.

We can now establish the fundamental relationship between direct and continuation semantics. First note that ΓH is actually a family of Γ-algebras depending upon the triple $\langle S, \mathscr{F}, \mathscr{B}\rangle$, which we call a *direct interpretation*. Similarly ΓW is actually a family depending upon the quadruple $\langle S, O, \mathscr{G}, \mathscr{B}\rangle$, which we call a *continuation interpretation*. (This kind of dependence will hold for all of our target algebras describing direct or continuation semantics.)

Consider a direct interpretation $\langle S, \mathscr{F}, \mathscr{B}\rangle$ and a continuation interpretation $\langle S, O, \mathscr{G}, \mathscr{B}\rangle$ with the same S and $\mathscr{B}$. Let $\alpha \in H \to W$ be the function such that $\alpha(h) = \lambda c.\ c * h$. If $\mathscr{G} = \alpha\cdot\mathscr{F}$, then the continuation interpretation is said to be an *image* of the direct interpretation. If in addition $O = S_\perp$, then it is an *exact* image. Each direct interpretation obviously has many images, one of which is exact. There are, however, continuation interpretations (where primitive instructions have "abnormal" meanings) which are not the image of any direct interpretation. Then:

PROPOSITION 1. *If the continuation interpretation is an image of the direct interpretation, then α is a homomorphism from ΓH to ΓW. If the image is exact, then a left inverse for α is provided by the function $\beta \in W \to H$ such that $\beta(w) = w(J_S)$.*

PROOF. The reader may verify that α satisfies the equations for a homomorphism from ΓH to ΓW and that it is strict and continuous. In the exact case, $\beta(\alpha(h)) = \alpha(h)(J_S) = J_S * h = h$. $\square$

Thus we have the following diagram of homomorphisms:

$$\begin{tikzcd}
 & \Gamma H \arrow[dd, dashed, "\alpha"] \\
\Gamma\mathrm{In}\Gamma \arrow[ur, "\mu_{\Gamma H}"] \arrow[dr, "\mu_{\Gamma W}"'] & \\
 & \Gamma W
\end{tikzcd} \qquad \text{(I)}$$

where α occurs only if the continuation interpretation is an image of the direct interpretation. Since $\Gamma\mathrm{In}\Gamma$ is an initial algebra, which implies $\alpha\cdot\mu_{\Gamma H} = \mu_{\Gamma W}$, this diagram commutes, i.e. all paths with the same origin and destination denote equal compositions of functions.

In other words, when the continuation interpretation is an image of the direct interpretation, α maps the direct meaning of any decision table diagram into its continuation meaning. Moreover, when the image is exact, β maps the continuation meaning back into the direct meaning.

Processes

Let Δ be the signature such that $\Delta_{\langle\,\rangle} = \{I\}$ and $\Delta_T = F$. Then the domain of process components is $\text{In}\Delta$, and the domain of processes is $Z = T \to \text{In}\Delta$. Not only do we want to define the semantics of these entities, but also to connect this semantics with the semantics of decision table diagrams. For this purpose, we assume that the semantics of decision table diagrams is given by an arbitrary target algebra ΓX (of which ΓH and ΓW are instances), and we derive the semantics of processes and their components from ΓX.

Intuitively, in a Γ-algebra, ΓX_I means "Do nothing," $\Gamma X_f(x)$ means "Execute f and then execute x," and $\Gamma X_D(\mathbf{x})$ means "Compute a table t of the current values of Boolean expressions and then execute $\mathbf{x}(t)$." In a Δ-algebra, ΔX_I means "Do nothing," and $\Delta X_f(\mathbf{x})$ means "Execute f, compute a table t of the current values of Boolean expressions, and then execute $\mathbf{x}(t)$." This suggests that, for any Γ-algebra ΓX, we define ΔX to be the Δ-algebra with the same carrier such that

$$\Delta X_I = \Gamma X_I, \quad \Delta X_f(\mathbf{x}) = \Gamma X_f(\Gamma X_D(\mathbf{x})).$$

Now suppose that $z \in Z$ is a process. Then the meaning of a process component $z(t)$ will be $\mu_{\Delta X}(z(t))$. But the meaning of z itself is "Compute a table t of the current values of Boolean expressions and then execute the meaning of $z(t)$," or, in other words, ΓX_D of the function of t which gives the meaning of $z(t)$. Thus processes are mapped into their meanings by the function $\delta_{\Gamma X} \in Z \to X$ such that

$$\delta_{\Gamma X}(z) = \Gamma X_D(\lambda t.\ \mu_{\Delta X}(z(t))) = \Gamma X_D(\mu_{\Delta X} \cdot z).$$

The previously given semantic equations for processes are simply assertions that the direct and continuation semantics of processes are given by $\delta_{\Gamma H}$ and $\delta_{\Gamma W}$ and the direct and continuation semantics of process components are given by $\mu_{\Delta H}$ and $\mu_{\Delta W}$.

The δ-functions bear the following relationship to homomorphisms between Γ-algebras:

PROPOSITION 2. *If $\rho \in \Gamma X \to \Gamma X'$, then $\rho \cdot \delta_{\Gamma X} = \delta_{\Gamma X'}$.*

PROOF. Suppose $\rho \in \Gamma X \to \Gamma X'$. It is easily seen that this implies $\rho \in \Delta X \to \Delta X'$ (which is not surprising since the definition of ΔX in terms of ΓX is an instance of the standard notion of a "derived algebra"). But then $\rho \cdot \mu_{\Delta X} \in \Delta\text{In}\Delta \to \Delta X'$ must be the unique homomorphism $\mu_{\Delta X'}$. Thus $\rho(\delta_{\Gamma X}(z)) = \rho(\Gamma X_D(\mu_{\Delta X} \cdot z)) = \Gamma X'_D(\rho \cdot \mu_{\Delta X} \cdot z) = \Gamma X'_D(\mu_{\Delta X'} \cdot z) = \delta_{\Gamma X'}(z)$. □

Informally we motivated the definition of processes by suggesting that they were a kind of canonical form for decision table diagrams. If this suggestion is true, then it should be possible to translate decision table diagrams into processes. By regarding this translation as a kind of meaning, we can define it algebraically: We define a target algebra ΓZ such that decision table diagrams are translated into processes by the unique homomorphism $\mu_{\Gamma Z} \in \Gamma\text{In}\Gamma \to \Gamma Z$. It turns out that the right definition of ΓZ is the Γ-algebra with carrier Z such that

$$\begin{aligned}
&\Gamma Z_I = \lambda t.\ \Delta\text{In}\Delta_I,\\
&\Gamma Z_f(z) = \lambda t.\ \Delta\text{In}\Delta_f(z),\\
&\Gamma Z_D(\mathbf{z}) = \lambda t.\ \mathbf{z}(t)(t) \qquad (\text{where } \mathbf{z} \in T \to Z = T \to (T \to \text{In}\Delta)).
\end{aligned}$$

This definition leads to the following relationship between ΓZ and the δ-functions:

PROPOSITION 3. (1) *ΓZ is a key algebra. If it exists, the unique homomorphism from ΓZ to ΓX is $\delta_{\Gamma X}$.* (2) *$\delta_{\Gamma In\Gamma}$ is a right inverse for $\mu_{\Gamma Z}$.*

PROOF. The definition of ΓZ gives rise to the derived algebra ΔZ. By checking the appropriate homomorphic equations, the reader may verify that $\lambda x.\ \lambda t.\ x$ is a homomorphism from $\Delta\text{In}\Delta$ to ΔZ and is therefore equal to the unique homomorphism $\mu_{\Delta Z}$. This equality implies that $\delta_{\Gamma Z}$ is the identity function for Z since $\delta_{\Gamma Z}(z) = \Gamma Z_D(\mu_{\Delta Z} \cdot z) = \lambda t.\ (\mu_{\Delta Z} \cdot z)(t)(t) = \lambda t.\ \mu_{\Delta Z}(z(t))(t) = \lambda t.\ z(t) = z$.

To establish (1), let ρ be any homomorphism from ΓZ to ΓX. Then $\rho = \rho \cdot \delta_{\Gamma Z}$, and, by Proposition 2, $\rho \cdot \delta_{\Gamma Z} = \delta_{\Gamma X}$. To establish (2), take ρ in Proposition 2 to be $\mu_{\Gamma Z} \in \Gamma\text{In}\Gamma \to \Gamma Z$. Then $\mu_{\Gamma Z} \cdot \delta_{\Gamma\text{In}\Gamma} = \delta_{\Gamma Z} = I_Z$. □

Although ΓZ is a key algebra, it is not initial since it is not isomorphic to $\Gamma\mathrm{In}\Gamma$. In other words, there are Γ-algebras for which there is no homomorphism from ΓZ. However, we are really only interested in target algebras in which D behaves like a decision operation. This behavior is characterized by the following *decision laws*, which are the algebraic formulation of the equivalences for D-operations given earlier:

(1) $(\forall x \in X) \quad \Gamma X_D(\lambda t.\, x) = x$;
(2) $(\forall g \in T \to (T \to X)) \quad \Gamma X_D(\lambda t.\, \Gamma X_D(\lambda t'.\, g(t, t'))) = \Gamma X_D(\lambda t.\, g(t, t))$.

The following proposition shows that ΓZ is initial in the restricted class of Γ-algebras which obey the decision laws. The reader may verify that this class includes ΓH, ΓW, and ΓZ, but not $\Gamma\mathrm{In}\Gamma$.

Proposition 4. *If* ΓX *obeys the decision laws, then* $\delta_{\Gamma Z} \in \Gamma Z \to \Gamma X$.

Proof. The necessary homomorphic equations are established by:

$$\begin{aligned}
\delta_{\Gamma X}(\Gamma Z_I) &= \Gamma X_D(\mu_{\Delta X}\cdot(\lambda t.\, \Delta\mathrm{In}\Delta_I)) = \Gamma X_D(\lambda t.\, \mu_{\Delta X}(\Delta\mathrm{In}\Delta_I))\\
&= \mu_{\Delta X}(\Delta\mathrm{In}\Delta_I) = \Delta X_I = \Gamma X_I;\\
\delta_{\Gamma X}(\Gamma Z_f(z)) &= \Gamma X_D(\mu_{\Delta X}\cdot(\lambda t.\, \Delta\mathrm{In}\Delta_f(z)))\\
&= \Gamma X_D(\lambda t.\, \mu_{\Delta X}(\Delta\mathrm{In}\Delta_f(z))) = \mu_{\Delta X}(\Delta\mathrm{In}\Delta_f(z)) = \Delta X_f(\mu_{\Delta X}\cdot z)\\
&= \Gamma X_f(\Gamma X_D(\mu_{\Delta X}\cdot z)) = \Gamma X_f(\delta_{\Gamma X}(z));\\
\delta_{\Gamma X}(\Gamma Z_D(\mathbf{z})) &= \Gamma X_D(\mu_{\Delta X}\cdot(\lambda t.\, \mathbf{z}(t)(t))) = \Gamma X_D(\lambda t.\, \mu_{\Delta X}(\mathbf{z}(t)(t)))\\
&= \Gamma X_D(\lambda t.\, \Gamma X_D(\lambda t'.\, \mu_{\Delta X}(\mathbf{z}(t)(t')))) = \Gamma X_D(\lambda t.\, \Gamma X_D(\mu_{\Delta X}\cdot \mathbf{z}(t)))\\
&= \Gamma X_D(\lambda t.\, \delta_{\Gamma X}(\mathbf{z}(t))) = \Gamma X_D(\delta_{\Gamma X}\cdot \mathbf{z}).
\end{aligned}$$

Strictness is established by

$$\delta_{\Gamma X}(\bot) = \Gamma X_D(\mu_{\Delta X}\cdot \bot) = \Gamma X_D(\lambda t.\, \mu_{\Delta X}(\bot)) = \mu_{\Delta X}(\bot) = \bot. \qquad \square$$

At this stage, we have the following diagram of homomorphisms:

$$\begin{tikzcd}
& & \Gamma H \arrow[dd, dashed, "a"] \\
\Gamma\mathrm{In}\Gamma \arrow[r, "\mu_{\Gamma Z}"] \arrow[urr, bend left, "\mu_{\Gamma H}"] \arrow[drr, bend right, "\mu_{\Gamma W}"'] & \Gamma Z \arrow[ur, "\delta_{\Gamma H}"] \arrow[dr, "\delta_{\Gamma W}"'] & \\
& & \Gamma W
\end{tikzcd} \qquad \text{(II)}$$

Since $\Gamma\mathrm{In}\Gamma$ and ΓZ are both key algebras, the diagram commutes.

Composers and Monoids

To treat linear and general flow diagrams, we will derive Λ-algebras from the Γ-algebras and then derive Ω-algebras from the Λ-algebras. In the final stage, however, in order to define the semicolon operation for Ω-algebras, we will need appropriate composition functions, or *composers*, for each of our algebras. We will introduce these composers as we go along and show at each stage that the homomorphisms we use remain valid when each algebra is augmented by adding its composer as an additional binary operation.

In general, when ΣX is a Σ-algebra with carrier X and $\Sigma_{\{\}}$ contains the constant I, we write $Fn\Sigma X$ for the Σ-algebra with carrier $X \to X$ such that $Fn\Sigma X_I = I_X$, and for $\sigma \in \Sigma_S$, $\sigma \neq I$, and $\mathbf{g} \in S \to (X \to X)$.

$$Fn\Sigma X_\sigma(\mathbf{g}) = \lambda x.\, \Sigma X_\sigma(\lambda s.\, \mathbf{g}(s)(x)).$$

When $S = \{1, \ldots, n\}$, the conventional form of the last equation is:

$$Fn\Sigma X_\sigma(g_1, \ldots, g_n) = \lambda x.\, \Sigma X_\sigma(g_1(x), \ldots, g_n(x)).$$

A homomorphism $\eta \in \Sigma X \rightarrow Fn\Sigma X$ is said to be a *composer* for ΣX.

For any signature Σ, we write $\Sigma^{;}$ to indicate the signature obtained from Σ by adding ; as a binary operator. When η is a composer for ΣX, we write $\Sigma^{;}X$ to denote the algebra obtained from ΣX by interpreting ; as the binary operation $\Sigma^{;}X_{;} = \eta$. (There is a potential ambiguity here which we avoid by never specifying more than one composer for the same algebra.)

For example, $Fn\Gamma H$ is the Γ-algebra with carrier $H \rightarrow H$ such that

$$Fn\Gamma H_I = I_H,$$
$$Fn\Gamma H_f(g) = \lambda h.\ \Gamma H_f(g(h)) = \lambda h.\ g(h)*\mathscr{F}(f),$$
$$Fn\Gamma H_D(g) = \lambda h.\ \Gamma H_D(\lambda t.\ g(t)(h)) = \lambda h.\ \lambda s.\ g(\lambda b.\ \mathscr{B}(b, s), h, s).$$

It is easily seen that the function $\lambda h_1.\ \lambda h_2.\ h_2*h_1$ is a homomorphism from ΓH to $Fn\Gamma H$ and is therefore a composer for ΓH. Thus $\Gamma^{;}H$ is obtained from ΓH by interpreting the semicolon as $\Gamma^{;}H_{;}\ (h_1, h_2) = h_2*h_1$.

This construction has the following general properties:

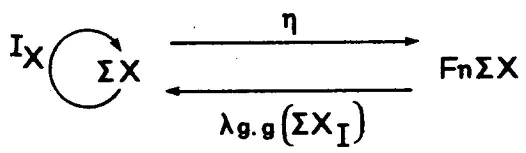
THEOREM 2. *If $I \in \Sigma_{()}$ and ΣX is a key Σ-algebra with a composer η, then $\Sigma^{;}X$ is a monoid, i.e.* (1) $(\forall x \in X)\ \eta(\Sigma X_I, x) = x$, (2) $(\forall x \in X)\ \eta(x, \Sigma X_I) = x$, (3) $(\forall x, y, z \in X)\ \eta(\eta(x, y), z) = \eta(x, \eta(y, z))$. *If in addition $\Sigma X'$ is a Σ-algebra with a composer and $\rho \in \Sigma X \rightarrow \Sigma X'$, then $\rho \in \Sigma^{;}X \rightarrow \Sigma^{;}X'$.*

PROOF. For a Σ-algebra ΣX and $x \in X$, we write $\Sigma X[x]$ to denote the algebra obtained from ΣX by reinterpreting I as the constant $\Sigma X[x]_I = x$. It is evident that $\rho \in \Sigma X \rightarrow \Sigma X'$ implies $\rho \in \Sigma X[x] \rightarrow \Sigma X'[\rho(x)]$. Also the definition of $Fn\Sigma X$ implies that $\lambda g.\ g(x) \in Fn\Sigma X \rightarrow \Sigma X[x]$.

Now suppose ΣX is a key algebra with the composer $\eta \in \Sigma X \rightarrow Fn\Sigma X$. Then:

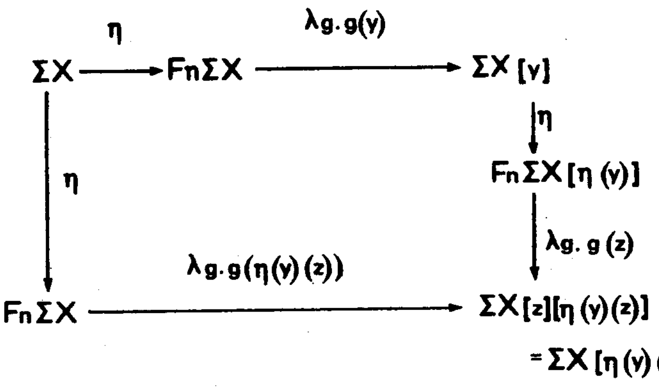
(1) $\eta(\Sigma X_I, x) = Fn\Sigma X_I(x) = x$.

(2) The assumption that ΣX is a key algebra implies that the diagram


$$\begin{tikzcd}
\Sigma X \arrow[loop left, "I_X"] \arrow[r, shift left, "\eta"] & Fn\Sigma X \arrow[l, shift left, "\lambda g.g(\Sigma X_I)"]
\end{tikzcd}$$

of homomorphisms commutes. Applying both compositions to x gives $\eta(x, \Sigma X_I) = x$.

(3) The following diagram also commutes:


$$\begin{tikzcd}
\Sigma X \arrow[r, "\eta"] \arrow[dd, "\eta"] & Fn\Sigma X \arrow[r, "\lambda g.g(y)"] & \Sigma X[y] \arrow[d, "\eta"] \\
 & & Fn\Sigma X[\eta(y)] \arrow[d, "\lambda g.g(z)"] \\
Fn\Sigma X \arrow[rr, "\lambda g.g(\eta(y)(z))"] & & \Sigma X[z][\eta(y)(z)] \\
 & & = \Sigma X[\eta(y)(z)]
\end{tikzcd}$$

Applying both compositions to x gives $\eta(\eta(x, y), z) = \eta(x, \eta(y, z))$.

If $\Sigma X'$ has the composer η' and $\rho \in \Sigma X \rightarrow \Sigma X'$, then the following diagram also commutes:

$$\begin{tikzcd}
\Sigma X \arrow[r, "\eta"] \arrow[d, "\rho"] & Fn\Sigma X \arrow[r, "\lambda g.\, g(y)"] & \Sigma X[y] \arrow[d, "\rho"] \\
\Sigma X' \arrow[r, "\eta'"] & Fn\Sigma X' \arrow[r, "\lambda g'.\, g'(\rho(y))"] & \Sigma X'[\rho(y)]
\end{tikzcd}$$

Applying both compositions to x gives $\rho(\eta(x, y)) = \eta'(\rho(x), \rho(y))$, which is the extra homomorphic equation needed to show $\rho \in \Sigma^{:}X \to \Sigma^{:}X'$. □

Our immediate goal is to extend Diagram (II) to the signature $\Gamma^{:}$. Suppose that we can provide a composer for each of the algebras in this diagram and that we can show that $\alpha \in \Gamma^{:}H \to \Gamma^{:}W$. Then, since every other homomorphism in (II) has the form $\rho \in \Gamma X \to \Gamma X'$ where ΓX is a key algebra (either $\Gamma\mathrm{In}\Gamma$ or ΓZ), Theorem 2 gives $\rho \in \Gamma^{:}X \to \Gamma^{:}X'$. Thus (II) will remain a diagram of homomorphisms if we change the signature of each algebra to $\Gamma^{:}$, i.e. if we add the appropriate composer to each algebra as a binary operation. Of course the diagram will continue to commute since the functions remain unchanged.

By deriving the appropriate algebras, verifying the resulting homomorphic equations, and checking strictness, one can show:

PROPOSITION 5. (1) $\lambda h_1.\ \lambda h_2.\ h_2 * h_1$ *is a composer for* ΓH. (2) $\lambda w_1.\ \lambda w_2.\ w_1 \cdot w_2$ *is a composer for* ΓW. (3) *If the continuation interpretation is an image of the direct interpretation, then* $\alpha \in \Gamma^{:}H \to \Gamma^{:}W$.

Moreover, a composer for $\Gamma\mathrm{In}\Gamma$ is automatically provided by initiality. In general:

PROPOSITION 6. *For any signature* Σ, $\mu_{Fn\Sigma In\Sigma}$ *is the (unique) composer for* $\Sigma In\Sigma$.

To complete our chain of reasoning, we must find a composer for ΓZ. To do so, we use an analogue of the idea of a Herbrand interpretation: We define a particular continuation semantics in which continuations are processes, and the meaning of a process is a function which forms the composition of that process with another.

Let ΓW^0 be the special case of ΓW for the continuation interpretation in which $S = T$, $O = \mathrm{In}\Delta$, $\mathscr{G} = \lambda f.\ \Gamma Z_f$, and $\mathscr{B} = \lambda b.\ \lambda t.\ t(b)$, so that $C = Z$, $W = Z \to Z$, and

$$\begin{aligned}
&\Gamma W^0_I = I_Z, \\
&\Gamma W^0_f(w) = \Gamma Z_f \cdot w, \\
&\Gamma W^0_D(\mathbf{w}) = \lambda z.\ \Gamma Z_D(\lambda t.\ \mathbf{w}(t, z)).
\end{aligned}$$

Then ΓW^0 is identical with $Fn\Gamma Z$. Thus:

PROPOSITION 7. $\delta_{\Gamma W^0}$ *is the (unique) composer for* ΓZ.

A side benefit is the right identity law for the monoid $\Gamma^{:}Z$, $\delta_{\Gamma W^0}(z)(\Gamma Z_I) = z$, which implies:

PROPOSITION 8. $\lambda w.\ w(\Gamma Z_I) \in (Z \to Z) \to Z$ *is a left inverse for* $\delta_{\Gamma W^0}$.

Thus processes are canonical for continuation semantics, since ΓW^0 is a continuation semantics which always gives distinct meanings to distinct processes.

In contrast there is no direct semantics with this property. For example, the distinct processes $\perp$ and $\Gamma Z_f(\perp)$ both mean $\perp$ in any direct semantics. Thus direct semantics induces a coarser relation of equivalence for processes (or other languages) than continuation semantics. This situation coincides with the author's intuition about the difference between direct and continuation semantics. It is one of the main reasons for not using a complete-lattice formalism; we have not been able to find a simple complete-lattice formulation of direct semantics such that $\perp$ and $\Gamma Z_f(\perp)$ have the same meaning in all direct and continuation semantics.

Linear Flow Diagrams

Linear flow diagrams are the initial algebra with the signature Λ such that $\Lambda_{(\,)} = \{I\}$, $\Lambda_{(1)} = F$, and $\Lambda_{(1,2)} = B$. Intuitively, in a Λ-algebra, I and each f have the same meaning as in the corresponding Γ-algebra, while $\Lambda X_b(x_1, x_2)$ means "Compute the current value of b and then execute x_1 if this value is true or x_2 if it is false." Thus, for any Γ-algebra ΓX, we define ΛX to be the Λ-algebra with the same carrier such that

$$\begin{aligned} &\Lambda X_I = \Gamma X_I, \\ &\Lambda X_f = \Gamma X_f, \\ &\Lambda X_b(x_1, x_2) = \Gamma X_D\,(\lambda t.\ cond_X(t(b), x_1, x_2)). \end{aligned}$$

Then:

Proposition 9. (1) *If* $\rho \in \Gamma X \to \Gamma X'$, *then* $\rho \in \Lambda X \to \Lambda X'$. (2) *A composer for* ΓX *is also a composer for* ΛX. (3) *If* $\rho \in \Gamma^{\cdot} X \to \Gamma^{\cdot} X'$, *then* $\rho \in \Lambda^{\cdot} X \to \Lambda^{\cdot} X'$.

The tedious but straightforward proof is left to the reader. It should be noted, however, that these results are more than an application of the standard notion of a derived algebra since they depend upon properties of the conditional function and the strictness of homomorphisms.

As a consequence (II) remains a diagram of homomorphisms if we change the signature of all algebras to either Λ or $\Lambda^{\cdot}$. For Λ we can add the initial algebra $\Lambda\text{In}\Lambda$ and its unique homomorphisms to the other algebras:

$$\begin{tikzcd}
 & & & \Lambda H \arrow[ddd, dashed, "a"] \\
 & & & \\
\Lambda\text{In}\Lambda \arrow[r, "\mu_{\Lambda\text{In}\Gamma}"] \arrow[urrr, bend left=40, "\mu_{\Lambda H}"] \arrow[rr, bend left=30, "\mu_{\Lambda Z}"] \arrow[drrr, bend right=40, "\mu_{\Lambda W}"'] & \Lambda\text{In}\Gamma \arrow[r, "\mu_{\Gamma Z}"] \arrow[urr, bend left=20, "\mu_{\Gamma H}"] \arrow[drr, bend right=20, "\mu_{\Gamma W}"'] & \Lambda Z \arrow[ur, "\delta_{\Gamma H}"] \arrow[dr, "\delta_{\Gamma W}"'] & \\
 & & & \Lambda W
\end{tikzcd} \qquad \text{(III)}$$

But Proposition 6 provides a composer for the added algebra, and Theorem 2 ensures that the added homomorphisms extend to $\Lambda^{\cdot}$. Thus (III) remains a diagram of homomorphisms if we change each signature to $\Lambda^{\cdot}$.

There is no right inverse for $\mu_{\Lambda\text{In}\Gamma}$ since there are decision table diagrams which cannot be mapped into any flow diagram with the same meaning in all semantics. A simple example is the decision table diagram

$$D[\lambda t.\ (f_1;\ cond_{\text{In}\Gamma}(t(b), (f_2; I), (f_3; I)))].$$

To execute this diagram, one must save a copy of the initial state during the execution of f_1 and then evaluate b in the saved state to determine whether f_2 or f_3 is to be executed next. One cannot evaluate b before executing f_1 since b may be nonterminating and f_1 may be an abnormal instruction.

A more spectacular example, for which the initial state must be saved indefinitely, is the decision table diagram $D[\lambda t.\ \theta(0, t)]$, where $\theta \in \text{Int} \to (T \to \text{In}\Gamma)$ is the least solution of

$$\theta(n, t) = cond_{\text{In}\Gamma}(t(b_n), (f_1;\ \theta(n+1, t)), (f_2;\ \theta(n+1, t)))$$

and $\{b_0, b_1, \ldots\}$ is some enumeration of B.

The existence of such decision table diagrams is regrettable, since their execution requires a kind of wholesale state-saving that is not in the spirit of imperative programming languages. It is an open question whether our development could be carried out with some more restricted notion of decision table diagram (and of process) such that every decision table diagram is equivalent to some flow diagram.

General Flow Diagrams

General flow diagrams are the initial algebra with the signature Ω such that $\Omega_{(\,)} = \{I\} \cup F$ and $\Omega_{(1,2)} = \{;\} \cup B$. For any $\Lambda^{;}$-algebra $\Lambda^{;}X$, we define ΩX to be the Ω-algebra with the same carrier such that

$$\Omega X_I = \Lambda^{;}X_I, \quad \Omega X_f = \Lambda^{;}X_f(\Lambda^{;}X_I), \quad \Omega X_; = \Lambda^{;}X_;, \quad \Omega X_b = \Lambda^{;}X_b.$$

Note that $\Lambda^{;}$ is used, rather than Λ, to provide an interpretation of the semicolon operator—we are finally letting our composers perform publicly. This is a standard instance of a derived algebra, so that $\rho \in \Lambda^{;}X \rightarrow \Lambda^{;}X'$ implies $\rho \in \Omega X \rightarrow \Omega X'$.

Thus, since the functions in Diagram (III) are homomorphisms of $\Lambda^{;}$-algebras, they are also homomorphisms of Ω-algebras. By adding the initial Ω-algebra and its unique homomorphisms to the other algebras, we obtain:

$$\begin{tikzcd}
& & & & \Omega H \arrow[dd, dashed, "a"] \\
\Omega\mathrm{In}\Omega \arrow[r, "\mu_{\Omega\mathrm{In}\Lambda}"] \arrow[urrrr, bend left, "\mu_{\Omega H}"] \arrow[rrr, bend left, "\mu_{\Omega Z}"] \arrow[rr, bend right, "\mu_{\Omega\mathrm{In}\Gamma}"'] \arrow[drrrr, bend right, "\mu_{\Omega W}"'] & \Omega\mathrm{In}\Lambda \arrow[r, "\mu_{\Lambda\mathrm{In}\Gamma}"] \arrow[urrr, bend left, "\mu_{\Lambda H}"] \arrow[rr, bend left, "\mu_{\Lambda Z}"] \arrow[drrr, bend right, "\mu_{\Lambda W}"'] & \Omega\mathrm{In}\Gamma \arrow[r, "\mu_{\Gamma Z}"] \arrow[urr, bend left, "\mu_{\Gamma H}"] \arrow[drr, "\mu_{\Gamma W}"'] & \Omega Z \arrow[ur, "\delta_{\Gamma H}"] \arrow[dr, "\delta_{\Gamma W}"'] & \\
& & & & \Omega W
\end{tikzcd} \qquad \text{(IV)}$$

Since $\Omega\mathrm{In}\Omega$ is initial, the diagram continues to commute.

An implication is that $\mu_{\Omega\mathrm{In}\Lambda}$ is a meaning preserving function from general to linear flow diagrams, which is slightly surprising since, intuitively, general flow diagrams really seem to be more general than linear flow diagrams. In fact this is a valid intuition, but in a more subtle sense: Call an element of an algebra *equational* if it is the least solution of a finite set of finite equations whose right sides are constructed from the operators of the algebra (as in [4, Sec. 5.1]). Then there are equational elements of $\Omega\mathrm{In}\Omega$, such as the least solution of $x = b \rightarrow ((f; x); g), I$, which are mapped by $\mu_{\Omega\mathrm{In}\Lambda}$ into nonequational elements of $\Lambda\mathrm{In}\Lambda$. Essentially, the equational elements of a Ω-algebra can express recursion, while those of a Λ-algebra can only express iteration.

Finally, we can "redefine" a Λ-algebra in terms of an Ω-algebra. For any Ω-algebra ΩX (actually, we are only interested in $\Omega\mathrm{In}\Omega$ and $\Omega\mathrm{In}\Lambda$), let ΛX be the Λ-algebra with the same carrier such that

$$\Lambda X_I = \Omega X_I, \quad \Lambda X_f(x) = \Omega X_;(\Omega X_f, x), \quad \Lambda X_b = \Omega X_b.$$

Again this is a standard instance of a derived algebra, so that $\rho \in \Omega X \rightarrow \Omega X'$ implies $\rho \in \Lambda X \rightarrow \Lambda X'$.

By applying this fact to $\mu_{\Omega\mathrm{In}\Lambda}$ and adding the initial Λ-algebra and its unique homomorphism into $\Lambda\mathrm{In}\Omega$, we get

$$\Lambda\mathrm{In}\Lambda \xrightarrow{\mu_{\Lambda\mathrm{In}\Omega}} \Lambda\mathrm{In}\Omega \xrightarrow{\mu_{\Omega\mathrm{In}\Lambda}} \Lambda\mathrm{In}\Lambda',$$

where the algebra on the left is the initial Λ-algebra and the algebra on the right is obtained from $\Omega\mathrm{In}\Lambda$ by the above definition. But in fact these two algebras are the same, as can be seen by comparing their operations. The only nontrivial case occurs for the operator f, where

$$\begin{aligned}\Lambda\mathrm{In}\Lambda'_f(x) &= \Omega\mathrm{In}\Lambda_;(\Omega\mathrm{In}\Lambda_f, x)\\ &= \Lambda{:}\mathrm{In}\Lambda_;(\Lambda{:}\mathrm{In}\Lambda_f(\Lambda{:}\mathrm{In}\Lambda_I), x)\\ &= \mu_{Fn\Lambda\mathrm{In}\Lambda}(\Lambda\mathrm{In}\Lambda_f(\Lambda\mathrm{In}\Lambda_I))(x)\\ &= Fn\Lambda\mathrm{In}\Lambda_f(Fn\Lambda\mathrm{In}\Lambda_I)(x)\\ &= (\lambda x.\ \Lambda\mathrm{In}\Lambda_f(Fn\Lambda\mathrm{In}\Lambda_I(x)))(x)\\ &= (\lambda x.\ \Lambda\mathrm{In}\Lambda_f(I_{\mathrm{In}\Lambda}(x)))(x) = \Lambda\mathrm{In}\Lambda_f(x).\end{aligned}$$

Then, since $\Lambda\mathrm{In}\Lambda$ is initial, we have $\mu_{\Omega\mathrm{In}\Lambda}\cdot\mu_{\Lambda\mathrm{In}\Omega} = \mu_{\Lambda\mathrm{In}\Lambda} = I_{\mathrm{In}\Lambda}$, i.e.

PROPOSITION 10. *$\mu_{\Lambda In\Omega}$ is a right inverse for $\mu_{\Omega In\Lambda}$.*

(But not a left inverse, since $\mu_{\Omega\mathrm{In}\Lambda}$ is a many-one function from general to linear flow diagrams.)

Conclusions

A variety of relationships among our languages and their semantics is specified by the commutativity of Diagram (IV), along with Propositions 1, 3(2), 8, and 10. Most of this information can be summarized as follows:

(1) The following are continuous functions among general flow diagrams ($\mathrm{In}\Omega$), linear flow diagrams ($\mathrm{In}\Lambda$), decision table diagrams ($\mathrm{In}\Gamma$), and processes (Z), which preserve meaning in any direct or continuation semantics:

$$\begin{tikzcd}\mathrm{In}\Omega \arrow[r, shift left, "\mu_{\Omega\mathrm{In}\Lambda}"] & \mathrm{In}\Lambda \arrow[l, shift left, "\mu_{\Lambda\mathrm{In}\Omega}"] \arrow[r, "\mu_{\Lambda\mathrm{In}\Gamma}"] & \mathrm{In}\Gamma \arrow[r, shift left, "\mu_{\Gamma Z}"] & Z. \arrow[l, shift left, "\delta_{\Gamma\mathrm{In}\Gamma}"]\end{tikzcd}$$

However, there is no such meaning-preserving function from decision table diagrams to linear flow diagrams.

(2) Continuation semantics subsumes direct semantics in the following sense: For any direct interpretation there is a continuation interpretation (its exact image) and functions $\alpha \in H \to W$ and $\beta \in W \to H$ such that α maps the direct meaning of any general flow diagram, linear flow diagram, decision table diagram, or process into the continuation meaning and β maps the continuation meaning back into the direct meaning. Direct semantics does not subsume continuation semantics in a similar sense.

(3) Processes are canonical for continuation, but not direct semantics, since there is a particular continuation interpretation (used to define ΓW^0) which always gives distinct meanings to distinct processes. There is no such direct semantics.

ACKNOWLEDGMENTS. The author would like to thank J.W. Thatcher, J.A. Goguen, and R. Milner for their encouraging and helpful comments on earlier drafts of this paper.

REFERENCES

1. BEKIC, H. Towards a mathematical theory of processes. Tech. Rep. TR25.125, IBM Vienna Lab., Vienna, Dec. 1971.
2. BURSTALL, R.M., AND LANDIN, P.J. Programs and their proofs: An algebraic approach. In *Machine Intelligence 4*, B. Meltzer and D. Michie, Eds., Edinburgh U. Press, Edinburgh, 1969, pp. 17–43.
3. EGLI, H., AND CONSTABLE, R.L. Computability concepts for programming language semantics. Proc. Seventh Annual ACM Symp. on Theory of Computing, May 1975, pp. 98–106; also *Theoretical Comptr. Sci. 2* (1976), 133–145.
4. GOGUEN, J.A., THATCHER, J.W., WAGNER, E.G., AND WRIGHT, J.B. Initial algebra semantics and continuous algebras. *J. ACM 24*, 1 (Jan. 1977), 68–95.
5. GORDON, M. Models of pure LISP. Experimental Programming Reports, No. 31, Dep. of Machine Intelligence, School of Artificial Intelligence, Edinburgh U., Edinburgh, 1973.
6. MILNER, R. Implementation and applications of Scott's logic for computable functions. Proc. ACM Conf. on Proving Assertions about Programs, SIGPLAN Notices (ACM) 7, 1 (Jan. 1972), 1–6.
7. MILNER, R. An approach to the semantics of parallel programs. Proc. Convegno di Informatica Teorica, Pisa, March 1973, pp. 283–302.
8. MORRIS, F.L. The next 700 programming-language decriptions. Unpublished.
9. PLOTKIN, G.D. A powerdomain construction. *SIAM J. Comptng. 5* (Sept. 1976), 452–487.

10. Reynolds, J.C. Notes on a lattice-theoretic approach to the theory of computation. Systems and Information Science, Syracuse U., Syracuse, N.Y., Oct. 1972.
11. Scott, D. Outline of a mathematical theory of computation. Proc. Fourth Annual Princeton Conf. on Inform. Sci. and Syst., 1970, pp. 169-176; also Tech. Monog. PRG-2, Programming Res. Group, Oxford U. Comptng. Lab., Oxford, 1970.
12. Scott, D. The Lattice of Flow Diagrams. Proc. Symp. on Semantics of Algorithmic Languages. E. Engeler, Ed., Springer Lecture Note Series No. 188, Springer-Verlag, Heidelberg, 1971, pp. 311-366; also, Tech. Monog. PRG-3, Programming Res. Group, Oxford U. Comptng. Lab., Oxford, Nov. 1970.
13. Scott, D. Continuous lattices. Proc. 1971 Dalhousie Conf., Springer Lecture Note Series No. 274, Springer-Verlag, Heidelberg, 1971, pp. 97-136; also, Tech. Monog. PRG-7, Programming Res. Group, Oxford U. Comptng. Lab., Oxford, Aug. 1971.
14. Scott, D., and Strachey, C. Towards a mathematical semantics for computer languages. Proc. Symp. on Computers and Automata, Polytechnic Institute of Brooklyn, Vol. 21, 1971, pp. 19-46; also, Tech. Monog. PRG 6, Oxford U. Comptng. Lab., Oxford, 1971.
15. Strachey, C., and Wadsworth, C.P. Continuations—a mathematical semantics for handling full jumps. Tech. Monog. PRG-11, Programming Res. Group, Oxford U. Comptng. Lab., Oxford, Jan. 1974.
16. Wand, M. On the recursive specification of data types. Category Theory Applied to Computation and Control, Lecture Notes in Computer Science, Vol. 25, Springer-Verlag, Berlin, 1975, pp. 214-217.

RECEIVED JULY 1975; REVISED NOVEMBER 1976