April 30, 1975

Backus

Class Notes III

Red Languages

A Red language L comprises a set of expressions E, a fixed expression called the store whose name is * [star], and a semantic function μ. The latter assigns a meaning, μe to those expressions e which correspond to a terminating computation. A particular Red language L is determined by (a) its set A of atoms, (b) its set F_L of primitive functions, (c) its representation function ρ which maps A onto F_L, and (d) its fixed expression store. The rest of the syntax and semantics of L ~~are~~ is determined by rules which apply to all Red languages.

Syntax of Red languages

(1) If e is an atom or if e = *, then e is an expression, and e has no components.

~~$e \in A$ or $e = *$ $\rightarrow$ $e \in E$~~

(2) If $e_1, \ldots, e_n$ are expressions, then $\langle e_1, \ldots, e_n \rangle$ is an expression called a sequence. e_k is its k^{th} member or component.

(3) If e and f are expressions, then (e : f) is an expression called an application. It has two components, e, its operator and f, its operand.

(4) Every expression is ~~formed~~ finitely generated by the use of rules (1) (2) and (3)

April 30, 1975

Backus

Class Notes III

Syntax, cont'd

An expression e' is a subexpression of e whenever e' = e or e' is a subexpression of a component of e. An expression c which does not have star or an application as a subexpression is a constant.

Example: $\langle (\langle A, B \rangle : C), * \rangle$

This is a sequence whose first member is an application and whose second is star. The operator of the application is a sequence of two atoms, etc.

Semantics

The primitive functions $f \in F_L$ of a Red language L map constants into expressions. Thus we can have functions like apply, where apply:$\langle x, y \rangle = (x:y)$, which map constants into non-constants. [In our earlier system, FP, apply would be meaningless, since on the left x must be an object and on the right, a function.]

Each primitive function f is represented by an primitive atom F, where $\rho F = f$. All other atoms represent the function δ whose value everywhere is the atom ϕ [meaning "unspecified"].

April 30, 1975

Backus

Class Notes III

Semantics, cont'd

The semantics of Red languages are so arranged that constant$_n$ sequences $\langle c_1, \ldots, c_n \rangle$ will represent functions in a manner analagous to the functional forms of FP, with the form being determined by c_1 with the other members being the parameters of the form. Thus if $c_1, \ldots, c_n$ represent the functions $f_1, \ldots, f_n$, we shall see that the primitive atom $\circ$ has the property that $\langle \circ, c_1, \ldots, c_n \rangle$ represents the FP function $f_1 \cdot f_2 \cdot \cdots \cdot f_n$ and so corresponds to the functional form for composition.

The store of a language L is normally a sequence whose members are cells. A cell is a sequence $\langle$CELL, name, contents$\rangle$ where CELL is an atom, and name and contents are expressions. The form $\langle$FETCH, X$\rangle$, when applied to such a store, will produce the contents of the first cell with name = X. The semantics of L permits a name^n_n to be used in place of a long sequence s which represents a useful function. If the first cell in the store with name n has contents s, ~~then~~ and n is a non-primitive atom [pn = δb], then (n:X) will have the same meaning as (s:X).

April 30, 1975
Backus

Class Notes III

Semantics, cont'd

Operational semantics

First we define the successor $\sigma(x:y)$ of an application $(x:y)$ whose components are both constants:

$$\sigma(x:y) \equiv \quad x \in A \;\&\; \rho x \neq \delta b \longrightarrow [\rho x]\, y\;;$$
$$x \in A \;\&\; \rho x = \delta b \longrightarrow ((\langle \text{FETCH}, x\rangle : *) : y)$$
$$x = \langle c_1, \ldots, c_n\rangle \longrightarrow (c_1 : \langle x, y\rangle)\;;\; \perp$$

Thus if x is a primitive atom $\sigma(x:y)$ is the result of applying the primitive function ρx to y. If x is a non-primitive atom, $\sigma(x:y)$ will ultimately cause the contents of the first cell in the store named x to be applied to y. We shall see later that the third rule allows the definition of functional forms.

To find μe, repeatedly modify e as follows until neither rule applies:

(1) Replace every occurrence of $*$ in e by the store, ~~then~~ until no occurrences remain. Then

(2) Find an innermost application $(x:y)$ in e and replace it by $\sigma(x:y)$

If this process terminates, the expression remaining is μe. If it does not, μe is undefined. [For example if the store is an expression with an occurrence of $*$ in it, then (1) is non-terminating.]

April 30, 1975
Backus

Class Notes III

Semantics, cont'd

Fixedpoint semantics

The semantic function μ defined above is the same as the least fixedpoint of the following functional τ:

$$[\tau\mu]e \equiv e = * \longrightarrow \mu\ \text{store}\ ;$$
$$e \in A \longrightarrow e\ ;$$
$$e = \langle e_1, \ldots, e_n \rangle \longrightarrow \langle \mu e_1, \ldots, \mu e_n \rangle$$
$$e = (x : y) \longrightarrow \mu\sigma(\mu x : \mu y)$$

A particular Red language

For this Red language L we shall take as our primitive functions those of FP on pp 4-5 of Class Notes I, Feb 19 1975 plus some others we shall need to recover the functional forms of FP. Thus the atom 1 will represent the selector function 1, and so on for 2, 3, ... TL will represent tl and in general the function whose FP-name is lower case xyz will be represented by capitalized atom XYZ. Thus for example

$$\text{rotl} : \langle A, B, C \rangle = \mu(\text{ROTL} : \langle A, B, C \rangle) = \langle B, C, A \rangle$$

April 30, 1975
Backus

Class Notes III

A Particular Red language, cont'd

We now show how FP's functional forms can be represented in L. Let compose be the following function, defined for any constant/object c,

$$\text{compose} : c \equiv c = \langle\langle c_0, c_1, \ldots, c_n\rangle, d\rangle \longrightarrow [n \geq 2 \longrightarrow (c_1 : (\langle c_0, c_2, \ldots, c_n\rangle : d)); n = 1 \longrightarrow (c_1 : d); \phi]; \phi$$

Let the atom ∘ [small circle] represent compose. If c_i represents the function f_i, then $\langle c_0, c_1, \ldots, c_n\rangle$ represents the function $f_1 \cdot f_2 \cdot \cdots \cdot f_n$, that is, $\mu(\langle c_0, c_1, \ldots, c_n\rangle : x) = f_1 \cdot f_2 \cdot \cdots \cdot f_n : x$.

First, this is true for $n = 1$ ✗ [we use arrows for transitions effected by the operational semantics]:

$$(\langle c_0, c_1\rangle : x) \longrightarrow (\circ : \langle\langle c_0, c_1\rangle, x\rangle) \longrightarrow \text{compose} : \langle\langle c_0, c_1\rangle, x\rangle \longrightarrow (c_1 : x) \longrightarrow f_1 : x$$

Assume it true for ~~$n \leq k$~~ $k < n$ for $n > 1$. Now consider

$$c = (\langle c_0, c_1, \ldots c_n\rangle : x) \longrightarrow (\circ : \langle\langle c_0, c_1, \ldots, c_n\rangle, x\rangle) \longrightarrow \text{compose} : \langle\langle c_0, c_1, \ldots, c_n\rangle, x\rangle \longrightarrow (c_1 : (\langle c_0, c_2, \ldots, c_n\rangle : x)) \longrightarrow \cdots \longrightarrow f_1 : \mu(\langle c_0, c_2, \ldots, c_n\rangle : x)$$

by induction we know that $\mu(\langle c_0 c_2, \ldots, c_n\rangle : x) = f_2 \cdot \cdots \cdot f_n : x$. Thus $\mu c = f_1 : [f_2 \cdot \cdots f_n] : x = f_1 \cdot f_2 \cdot \cdots \cdot f_n : x$.

April 30, 1975
Backus

Class Notes III

A particular Red language, cont'd

The bottom-up semantics of L force us to be very careful in defining the analog of $(p \longrightarrow f; g)$ lest we inadvertantly create a non-terminating application, say $(g:x)$ before finding that $(p:x)$ is T. We sketch a solution:

Let $\text{cond} : \langle x, y, z \rangle = \begin{cases} y & \text{if } x = T \\ z & \text{if } x = F \\ \phi & \text{otherwise} \end{cases}$

Let $\text{apply} : \langle x, y \rangle = (x:y)$

Let $\text{rearrg} : \langle \langle x_1, x_2, x_3, x_4 \rangle, y \rangle = \langle (x_2:y), \langle x_3, y \rangle, \langle x_4, y \rangle \rangle$

Let $\langle \text{FETCH}, \text{CN} \rangle : * = \langle 0, \text{APPLY}, \text{COND}, \text{REARRG} \rangle$

then $\langle \text{CN}, p, f, g \rangle$ represents the same function as $(p' \longrightarrow f'; g')$ where p, f, g represent p', f' and g'.