May 7 1975
Backus

# Class Notes IV

## Homework

Goal Define operator AA so that

$$\mu(\langle AA, c\rangle : \phi) = \phi$$
$$\mu(\langle AA, c\rangle : \langle x_1, \ldots, x_n\rangle) = \mu\langle (c:x_1), \ldots, (c:x_n)\rangle$$

Then $\langle AA, c\rangle$ will represent the FP function $\alpha f$ whenever $c$ represents $f$.

To define AA we will need the following Red operators for three FP functional forms: (1) composition, (2) n functions, and (3) condition

$$(1)\ \mu(\langle \circ, c_1, \ldots, c_n\rangle : x) = \mu(c_1 : (c_2 : (\cdots : (c_n : x)\cdots)))$$
$$(2)\ \mu(\langle NF, c_1, \ldots, c_n\rangle : x) = \mu\langle (c_1 : x), \ldots, (c_n : x)\rangle$$
$$(3)\ \mu(\langle CN, p, f, g\rangle : x) = \mu(f:x) \text{ if } \mu(p:x) = T$$
$$= \mu(g:x) \text{ if } \mu(p:x) = F$$
$$= \phi \text{ otherwise}$$

We shall use the FP notation as an abbreviation for each of these forms. Thus we shall write

| | |
|---|---|
| $c_1 \cdot c_2 \cdot \cdots \cdot c_n$ | for $\langle \circ, c_1, \ldots, c_n\rangle$ |
| $[c_1, \ldots, c_n]$ | for $\langle NF, c_1, \ldots, c_n\rangle$ |
| $(p \rightarrow f; g)$ | for $\langle CN, p, f, g\rangle$ |

[Notice that in Red languages each $c_i$, $p$, $f$, and $g$ here denote a constant, which in turn represents a function; whereas in FP forms each $c_i$, $p$, $f$, and $g$ stands for a function.]

May 7 1975
Backus

# Class Notes IV

Let

$$AA = (NULL \cdot 2 \longrightarrow 2\,;\ JN \cdot [f_1, f_2])$$

where

$$f_1 = APPLY \cdot [2 \cdot 1,\ 1 \cdot 2]$$
$$f_2 = APPLY \cdot [1,\ TL \cdot 2]$$

Then

$$f_1 : \langle\langle p, q\rangle, \langle x_1, \ldots, x_n\rangle\rangle = (q : x_1)$$
$$f_2 : \langle\langle p, q\rangle, x\rangle = (\langle p, q\rangle : (TL : x))$$

Thus

$$(\langle AA, c\rangle : x) \longrightarrow (AA : \langle\langle AA, c\rangle, x\rangle) \qquad \text{[by basic semantics]}$$

$x = \phi \longrightarrow \phi$ since $NULL \cdot 2 : \langle\langle\ \rangle, x\rangle = T$

$x \in A \longrightarrow \phi$ in Red we take $NULL : x = \phi$ instead of $\perp$

$$x = \langle x_1\rangle \longrightarrow (JN : \langle (C : x_1), (\langle AA, c\rangle : \phi)\rangle) \longrightarrow \langle (C : x_1)\rangle$$

Now assume $(\langle AA, c\rangle : \langle x_1, \ldots, x_n\rangle) \longrightarrow \alpha c : \langle x_1, \ldots, x_n\rangle$ for $n < K$ and $K > 1$. Then

$$x = \langle x_1, \ldots, x_K\rangle \longrightarrow JN : \langle (C : x_1), (\langle AA, c\rangle : \langle x_2, \ldots, x_n\rangle)\rangle$$

by induction then

$$\longrightarrow JN : \langle (C : x_1), \langle (C : x_2), \ldots, (C : x_n)\rangle\rangle = \langle (C : x_1), (C : x_2), \ldots, (C : x_n)\rangle$$

Thus for every $x$,

$$\mu(\langle AA, c\rangle : x) = \alpha c' : x$$

where $c$ represents the function $c'$.

Class Notes IK

May 7 1975
Backus

It is perhaps worth noting that had we been given AA as a primitive instead of NF, then NF can be defined as follows

$$NF = \langle AA, APPLY \rangle \cdot TL \cdot DISTR$$

Remarks

The exercises above in defining operators for various functional forms may create a misleading impression about Red languages and the proper style of Red programming. In FP we saw a little bit of the expressive power that can be achieved with a small fixed set of functional forms plus various functions. It is intended that Red programming would employ the same style as FP. The point of the above exercises, with their not-so-pleasant style, is simply to show that if a new functional form is found to be generally useful, it can be defined without adding to the basic facilities of Red languages or altering their semantics. [provided, of course, that an adequate set of primitive operators is available - and many small, but adequate sets exist.]

May 7 1975
Backus

# Class Notes IV

## Properties of Red Languages

### I An extended Church-Rosser property

Facts similar to the following ones, and others, are proved in a more general form by Paul McJones in "A Church-Rosser property of closed applicative languages." [a forthcoming IBM Research (San Jose) report available from the author.]

**Definition:** $e'$ is a **reduction** of $e$ iff $e'$ is obtainable from $e$ by (a) replacing an occurrence of $*$ by store or by (b) replacing an innermost star-free occurrence of an application $p$ by $\sigma p$.

**Definition** $e_1, \ldots, e_n$ is a **reduction sequence from** $e_1$ iff $e_{i+1}$ is a reduction of $e_i$ for $i = 1, \ldots, n-1$ and $e_n$ is a constant.

**Definition** The **order** $\theta(e)$ of $e$ is defined:

$$\theta(e) \equiv e \in A \longrightarrow 0\ ;\ e = * \longrightarrow \theta(\text{store}) + 1\ ;$$
$$e = \langle e_1, \ldots, e_n \rangle \longrightarrow \sum_{i=1}^{n} \theta(e_i)\ ;$$
$$e = (p:q) \longrightarrow 1 + \theta(p) + \theta(q) + \theta\sigma(\mu p : \mu q)$$

Thus $\theta(e)$ is the number of $*$'s and applications encountered in evaluating $e$.

May 7, 1975
Backus

# Class Notes IV

## An Extended Church-Rosser property, cont'd.

With the above definitions the following can be proved:

1) If $\mu e$ is defined [by fixedpoint semantics], then $\sigma(e) = n$ is defined.

2) If $\mu e$ is defined [fixedpoint], then every reduction sequence from e has length $\sigma(e)$ and ends with $\mu e$. If $\mu e$ is undefined, there is no reduction sequence from e.

Comment: This is the ~~extended~~ Church-Rosser property: the operational semantics always give the same result for e if they give one at all. The extend property is that if there is a result by any reduction sequence, then all will yield it and all in the same number of steps. Further, this result shows that the fixedpoint semantics and the operational semantics exactly agree.

3) If $\mu e$ is defined, then $\mu e = e$ iff e is a constant. [Thus C, the set of constants, is precisely the set of fixedpoints of $\mu$. And since $C \equiv \mu E$, $\mu$ is idempotent: $\mu \cdot \mu \equiv \mu$.]