Algebraic Effects and Handlers

1 The algebra stuff

Problem 1.1 (The theory of an associative unital operation)**.** Consider the theory T of an associative operation with a unit. It has a constant ϵ and a binary operation $\cdot$ satisfying equations

$$\begin{aligned}(x \cdot y) \cdot z &= x \cdot (y \cdot z)\\ \epsilon \cdot x &= x\\ x \cdot \epsilon &= x\end{aligned}$$

Give a useful description of the free model of T generated by a set X. You can either guess an explicit construction of free models and show that it has the required universal property, or you can analyze the free model construction (equivalence classes of well-founded trees) and provide a simple description of it.

Solution. I propose that, as with the semilattice example, the free model of T generated by a set X has the carrier set $\mathcal{P}_{<\omega}(X)$ of all finite subsets of X, with $[\![\epsilon]\!] = \emptyset$ and $[\![\cdot]\!] = \cup$. We have the same map $\eta : X \to \mathcal{P}_{<\omega}(X)$ defined by $x \mapsto \{x\}$.

Given any other model $(T, \epsilon, \cdot)$ and a map $f : X \to |T|$, we can define the homomorphism $\overline{f} : \mathcal{P}_{<\omega} \to |L|$ by

$$\overline{f}(\{x_1, x_2, \ldots, x_n\}) = f(x_1) \cdot f(x_2) \cdot \cdots \cdot f(x_n).$$

The diagram commutes because $\overline{f}(\eta(x)) = \overline{f}(\{x\}) = f(x)$. If g is another homomorphism for which the diagram commutes, we must have

$$\begin{aligned}g(\{x_1, \ldots, x_n\}) = g(\eta(x_1) \cup \cdots \cup \eta(x_n)) &= g(\eta(x_1)) \cdot \cdots \cdot g(\eta(x_n))\\ &= f(x_1) \cdot \cdots \cdot f(x_n) = \overline{f}(\{x_1, \ldots, x_n\}),\end{aligned}$$

so $\overline{f}$ is unique. □

Problem 1.2 (The theory of apocalypse)**.** We formulate an algebraic theory Time in it is possible to explicitly record passage of time. The theory has a single unary operation tick and no equations. Each application of tick records the passage of one time step.

Task: Give a useful description of the free model of the theory, generated by a set X.
Task: Let a given fixed natural number n be given. Describe a theory $\mathsf{Apocalypse}$ which extends the theory Time so that a computation crashes (aborts, necessarily terminates) if it performs more than n of ticks. Give a useful description of its free models.

Advice: do *not* concern yourself with any sort of operational semantics which somehow "aborts" after n ticks. Instead, use equations and postulate that certain computations are equal to an aborted one.

Solution. We observe that the set $\mathsf{Tree}_{\mathsf{Time}}(X)$ is the set of sticks with $\mathsf{return}\ x$ at their leaves and tick at each node. Because we have no equations, the free model generated by X is just the set of trees $\mathsf{Tree}_{\mathsf{Time}}(X)$, our map $\eta_X : X \to \mathsf{Tree}_{\mathsf{Time}}(X)$ maps x to $\mathsf{return}\ x$, and the interpretation of tick just adds one more tick to a tree. A nice way to represent the carrier set of the free model generated by X is as the set $X \times \mathbb{N}$ where the natural number (including 0) represents the number of ticks applied to a starting element in X. With this representation, the operation tick maps (x, n) to $(x, n+1)$.

The theory $\mathsf{Apocalypse}$ contains the equation $\mathsf{tick}^n(x) = \mathsf{tick}^{n+1}(x)$. In our tree repsesentation, we now have that all trees of height n or greater are equal under $\approx_{\mathsf{Apocalypse}}$. Note that we still differentiate between aborted computations that started in different places, which is just to make things simpler. This leads to the free model with carrier set $X \times \{0, \ldots, n\}$ and the operation

$$\mathsf{tick}((x, m)) = \begin{cases} (x, m+1) & \text{if } m < n \\ (x, m) & \text{if } m = n. \end{cases}$$

□

2 The language stuff

Problem 2.1 (Products)**.** Add simple products `A ×B` to the core language:

1. Extend the syntax of values with pairs.
2. Extend the syntax of computations with an elimination of pairs, e.g., `do (x,y) ←`$\mathtt{c}_1$`in` $\mathtt{c}_2$.
3. Extend the operational semantics.
4. Extend the typing rules.

Solution. We extend the syntax of values as

```
v ::= ...
    | (v1, v2)        (pair)
```

and the syntax of computations as

```
c ::= ...
    | do (x,y) ← c1 in c2    (pair elim)
```

Our operational semantics are extended with the following rule:

```

-------------------------------------------------------
do (x, y) ← return (v1, v2) in c2  ↦  c2[v1/x, v2/y]
```

The new definition for value type is:

```
          A, B := bool | A → C | C ⇒ D | A × B
```

We add one new rule for value typing:

```
Γ ⊢ x : A   Γ ⊢ y : B
--------------------
 Γ ⊢ (x, y) : A × B
```

and one new rule for computation typing:

```
Γ ⊢ c1 : (A1 × A2)!Δ Γ, x:A1, y:A2 ⊢c2 : B!Δ
-------------------------------------------
      Γ ⊢ (do (x, y) ← c1 in c2) : B!Δ
```

□

Problem 2.2 (Non-terminating program)**.** Define a program which prints infinitely many booleans. You may assume that the `print : bool →unit` operation is handled appropriately by the runtime environment. For extra credit, make it "funny".

Solution. I think this works?

```
fix loop .
    with handler {
        return x ↦ return x,
        print(x, _) ↦ print(x, λ_ . loop)
    } handle print(true,
        λ_ . solve_world_hunger((), λ_ . return true))
```

□