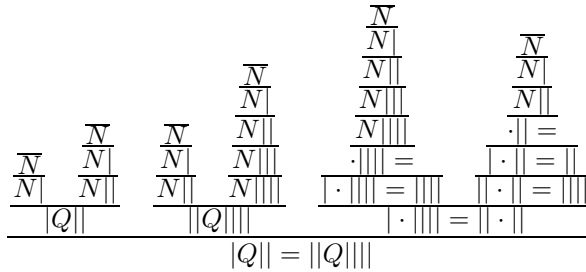


CSE 230 MIDTERM SOLUTIONS

Midterm of 7 February 2002

For reference, the midterm can be found at
<http://www-cse.ucsd.edu/~goguen/courses/230/230mid02.ps>.

1. (a) The system derives non-negative integers, quotients, equalities of non-negative integers, equalities of quotients, and so on. Idiosyncratically, the system allows quotients with denominator zero; in fact, it is possible to derive $n/0 = m/0$ for any non-negative integers m, n .
- (b)



2. There are two basic approaches to this problem. The first is to write idiomatic OBJ code for accomplishing the same thing, but in a different way; this is not very difficult. The second approach is to model the Post system rather exactly, with strings as the basic data type, and with each rule as an operation on strings; this is the most faithful to the question (but it runs incredibly slowly if `id: nil` is used instead of `idr: nil`). Here computations are sequences of deductions, i.e. proofs in the Post system. There is also a hybrid approach, which combines aspects other two, modeling rules as operations, but taking advantage of OBJ's power for defining data types; it runs quickly, and again, its computations are Post system proofs.

Code for the first approach

```
obj RAT is sorts Nat Rat .
  op N   : -> Nat .
  op _|_ : Nat -> Nat .
```

```

op _+_ : Nat Nat -> Nat .
op *_ : Nat Nat -> Nat .
op _Q_ : Nat Nat -> Rat .
op _==_ : Rat Rat -> Bool .

vars x y z w : Nat .
eq N + x = x .
eq x | + y = (x + y) | .
eq N * x = N .
eq x | * y = (x * y) + y .
eq x Q y == z Q w = x * w == y * z .
endo

red (N |) Q (N | |) == (N | |) Q (N | | | |) .

```

Code for the second approach

```

obj RatPost is sort St .
vars u v w x y z : St .

ops nil N | * e Q : -> St .
op __ : St St -> St [assoc id: nil].

op [1] : -> St .
eq [1] = N .

op [2]_ : St -> St .
eq [2](N x) = N x | .

op [4]_ : St -> St .
eq [4](N x) = * x e .

op [5]_ : St -> St .
eq [5](x * y e z) = (x | * y e z y) .

op [6]__ : St St -> St .
eq [6](N x) (N y) = (x Q y) .

op [7]___ : St St St -> St .
eq [7](w Q x) (y Q z) (w * z e x * y) = (w Q x e y Q z) .

op [8]__ : St St -> St .
eq [8](x * y e z) (u * v e z) = (x * y e u * v) .

endo

```

```

open .
let n0 = [1] .
let n1 = [2] n0 .
let n2 = [2] n1 .
let n3 = [2] n2 .
let n4 = [2] n3 .

let q1 = [6] n1 n2 .
let q2 = [6] n2 n4 .

let e1 = [5] [5] [4] n2 .
let e2 = [5] [4] n4 .
let e3 = [8] e2 e1 .
let e4 = [7] q1 q2 e3 .

red n4 .
red q2 .

red e1 .
red e2 .
red e3 .
red e4 .

close

```

Code for the third approach

```

obj RatPost is
  sorts Nat NatN Term Eq Rat .
  subsort Nat < Term .
  vars u v w x y z : Nat .

  ops nil | : -> Nat .
  op __ : Nat Nat -> Nat [assoc id: nil].
  op N_ : Nat -> NatN .
  op *_ : Term Term -> Term .
  op _e_ : Term Term -> Eq [comm].
  op _e_ : Rat Rat -> Eq .
  op _Q_ : Nat Nat -> Rat .

  op [1] : -> NatN .
  eq [1] = N nil .

  op [2]_ : NatN -> NatN .
  eq [2](N x) = N (x |).

```

```

op [4]_ : NatN -> Eq .
eq [4](N x) = (nil * x) e nil .

op [5]_ : Eq -> Eq .
eq [5]((x * y) e z) = (x | * y) e (z y).

op [6]__ : NatN NatN -> Rat .
eq [6](N x) (N y) = (x Q y).

op [7]___ : Rat Rat Eq -> Eq .
eq [7](w Q x) (y Q z) ((w * z) e (x * y)) = (w Q x) e (y Q z) .

op [8]__ : Eq Eq -> Eq .
eq [8]((x * y) e z) ((u * v) e z) = (x * y) e (u * v).

endo

open .
let n0 = [1] .
let n1 = [2] n0 .
let n2 = [2] n1 .
let n3 = [2] n2 .
let n4 = [2] n3 .

let q1 = [6] n1 n2 .
let q2 = [6] n2 n4 .

let e1 = [5] [5] [4] n2 .
let e2 = [5] [4] n4 .
let e3 = [8] e2 e1 .
let e4 = [7] q1 q2 e3 .

red n4 .
red q2 .

red e1 .
red e2 .
red e3 .
red e4 .

close

```

This code and the corresponding output can be found at
<http://www-cse.ucsd.edu/~goguen/courses/230/obj/ratpost/r.html>.

3. (a) “We take a view focusing on the underlying significance of the con-

structs. This perspective we call an *abstract syntax*.

“... Sometimes it is useful to describe a language in an *analytic* manner. This is an *abstract* syntax description. It is abstract in that it is independent of the notation used to represent the language constructs. ... Note that the level of abstractness is subjective. What is abstract for one purpose may be too concrete for another. Different levels of abstractness may be possible” (Stansifer, p. 68).

“Since abstract syntax is given by the term algebra with signature given by the grammar, we can get a denotational semantics by building a semantic algebra with the same signature, and letting the unique many sorted homomorphism from the term algebra to the semantic algebra provide the denotations of the terms, which essentially are parse trees” (class notes on chapter 8 of Stansifer).

See section 2.5.2 (pp. 68–69) in Stansifer for an example of the abstract syntax of `while` programs.

- (b) A *synthesized attribute* of a term is one whose value is determined from the constituent parts of the term. “An attribute grammar is a context-free grammar augmented by attributes and semantic rules” (Stansifer, p. 53). Though the term *synthesized attribute grammar* does not appear in the book, it is reasonable to presume it means an attribute grammar with only synthesized attributes.

Here is an attribute grammar for infix addition expressions, where E is an expression and $E.v$ is a synthesized attribute representing the value of E :

$$\begin{aligned} E_1 &\rightarrow E_2 + E_3 \\ E_1.v &:= E_2.v + E_3.v \end{aligned}$$

- 4. (a) In *infix syntax* operators are between their operands, as in `1 + 2`.
In *postfix syntax* operators are after their operands, as in `1 2 +`.
In *mixfix syntax* operators comprise multiple tokens intermingled with their operands, as in `add 1 and 2`.
- (b) *Dereferencing* an l-value yields the contents of the specified location. In ML, `!X` dereferences the l-value denoted by `X`.
- (c) “An *environment* is a mapping of names to values” (Stansifer, p. 89). If the name `x` maps to the value `1` under the environment ρ , then we write $\rho(x) = 1$.
The word *storage* is part of the term *storage semantics* in section 3.3.1 of Stansifer. When it is used as a storage model, a mapping of names to values yields storage semantics.

- (d) *Aliased variables* have the same l-value. For example, if call-by-reference is used, the procedure invocation $f(x, x)$ will cause the two formal parameters of f to be aliases.
 - (e) A *regular expression* denotes a language over an alphabet using five rules: empty, atom, alternation, concatenation, and closure. See section 2.2.2 (pp. 43–46) in Stansifer for more detail. An example of a regular expression is $((a \cdot b^*) + \emptyset)$.
5. (a) The standard model (or initial algebra or standard interpretation) A of the signature **NATLIST** consists of:
- All the elements of the standard **NAT** model, including the carrier set A_{Nat} , which has as many distinct elements as there are natural numbers.
 - The carrier set $A_{\text{List}} \supset A_{\text{Nat}}$, which has as many distinct elements as there are lists of natural numbers (including the empty list).
 - The nullary operation (or constant) A_{nil} , which returns one particular element of A_{List} we call the *empty list*.
 - The binary list concatenation operation A_c , whose domain is $A_{\text{List}} \times A_{\text{List}}$ and whose range is A_{List} . If either of its arguments is the empty list, it returns its other argument; else it returns an element of A_{List} which is neither argument.
 - The operation A_{head} , whose domain is A_{List} and whose range is A_{Nat} . For $N \in A_{\text{Nat}}$ and $L_1, L_2 \in A_{\text{List}}$, if $A_c(N, L_1) = L_2$ then $A_{\text{head}}(L_2) = N$.
 - The operation A_{tail} , whose domain is A_{List} and whose range is A_{List} . For $N \in A_{\text{Nat}}$ and $L_1, L_2 \in A_{\text{List}}$, if $A_c(N, L_1) = L_2$ then $A_{\text{tail}}(L_2) = L_1$.

Another model (or algebra or interpretation) A' of **NATLIST** might include *junk*: elements which cannot be represented by any **NATLIST**-term. For example, the carrier set A'_{List} could contain all the real numbers. Also, there might be ground equations true of A' which cannot be proved from the equations in **NATLIST**, such as $\text{tail nil} = \text{nil}$; for such unprovable equations to be true is called *confusion*. By definition, the standard model has no junk and no confusion.

- (b) To get the standard model in **OBJ**, replace **th** with **obj**. This works because objects have standard semantics.
- (c) We use **th** when we intend loose semantics: we don't really care what model is used so long as it satisfies the requirements in the signature. This is the case for stores; any such model will suffice. We use **obj** when we intend the standard model: everything specified in the signature is true and nothing more. We desire standard semantics for the language semantics, so we use **obj** for this.

In practice, **OBJ3** treats both kinds of signatures identically, so deciding between **th** and **obj** is really an expression of intent for the benefit of a person reading the code.