

Syntax: Object Algebras

April 2, 2015

1 Object Algebra Interface

1.1 Inheritance \times

1.1.1 Template

```
BEFORE: sig  $N_{AI}[\overline{T_{AI}}]$  where  $\overline{N_{CS} : T_{CS}};$ 
```

```
AFTER: type  $N_{AI}[\overline{T_{AI}}] = \{\overline{N_{CS} : T_{CS}}\};$ 
```

1.1.2 Example: ExpAlg[E]

```
BEFORE: sig ExpAlg[E] where  
    lit : Int -> E,  
    add : E -> E -> E;
```

```
AFTER: type ExpAlg[E] = {  
    lit : Int -> E,  
    add : E -> E -> E  
};
```

1.2 Inheritance $\sqrt{}$

1.2.1 Template

```
BEFORE: sig  $N_{AI}[\overline{T_{AI}}]$  extends  $\overline{N_{AI_2}[\overline{T_{AI_2}}]}$  where  $\overline{N_{CS} : T_{CS}};$ 
```

```
AFTER: type  $N_{AI}[\overline{T_{AI}}] = \&(\overline{N_{AI_2}[\overline{T_{AI_2}}]}) \& \{\overline{N_{CS} : T_{CS}}\};$ 
```

NB. How to avoid multiple inheritance from the same interface? Throw an exception if two records with same labels are combined with an “&”?

1.2.2 Example: StatAlg[E, S]

```
BEFORE: sig StatAlg[E, S] extends ExpAlg[E] where  
    seq : S -> S -> S,  
    asn : String -> E -> S;
```

```
AFTER: type StatAlg[E, S] = (ExpAlg[E]) & {  
    seq : S -> S -> S,  
    asn : String -> E -> S  
};
```

2 Object Algebra

2.1 Inheritance \times

2.1.1 Template

BEFORE: **algebra** N_A **implements** $\overline{N_{AI}[T_A]}$ **where** $\overline{t@}(N_{CS} \ \overline{x}) = \overline{E}$;

AFTER: **let** $N_A = \overline{[T_A/T_{AI}]}$ $\{N_{CS} = \lambda(\overline{x} : \overline{T_{CS}}). \{t = \overline{E}\}\}$;

2.1.2 Example: EvalExpAlg

BEFORE: **type** IEval = { eval : Int };
algebra EvalExpAlg **implements** ExpAlg[IEval] **where**
 eval@(lit x) = x,
 eval@(add x y) = x.eval + y.eval;

AFTER: **type** IEval = { eval : Int };
let EvalExpAlg = {
 lit = $\lambda(x : \text{Int}) \rightarrow \{ \text{eval} = x \}$,
 add = $\lambda(x : \text{IEval}) \rightarrow \lambda(y : \text{IEval}) \rightarrow \{ \text{eval} = x.\text{eval} + y.\text{eval} \}$
};

2.2 Inheritance $\sqrt{}$

2.2.1 Template

BEFORE: **algebra** N_A **extends** $\overline{N_{A_2}}$ **implements** $\overline{N_{AI}[T_A]}$ **where** $\overline{t@}(N_{CS} \ \overline{x}) = \overline{E}$;

AFTER: **let** $N_A = ((,.) (\overline{N_{A_2}})) \ , \ , \ \overline{[T_A/T_{AI}]}$ $\{N_{CS} = \lambda(\overline{x} : \overline{T_{CS}}). \{t = \overline{E}\}\}$;

NB. The same for multiple inheritance.

2.2.2 Example: PrintStatAlg

BEFORE: **type** IPrint = { print : String };
algebra PrintExpAlg **implements** ExpAlg[IPrint] **where**
 print@(lit x) = $\backslash\{x\}$,
 print@(add x y) = $\backslash\{x.\text{print}\} + \backslash\{y.\text{print}\}$;
algebra PrintStatAlg **extends** PrintExpAlg **implements** StatAlg[IPrint, IPrint] **where**
 print@(seq x y) = $\backslash\{x.\text{print}\} \ || \ \backslash\{y.\text{print}\}$,
 print@(asn x y) = $\backslash\{x\} = \backslash\{y.\text{print}\}$;

AFTER: **type** IPrint = { print : String };
let PrintExpAlg = {
 lit = $\lambda(x : \text{Int}) \rightarrow \{ \text{print} = \backslash\{x\} \}$,
 add = $\lambda(x : \text{IPrint}) \rightarrow \lambda(y : \text{IPrint}) \rightarrow \{ \text{print} = \backslash\{x.\text{print}\} + \backslash\{y.\text{print}\} \}$
};
let PrintStatAlg = PrintExpAlg , , {
 seq = $\lambda(x : \text{IPrint}) \rightarrow \lambda(y : \text{IPrint}) \rightarrow \{ \text{print} = \backslash\{x.\text{print}\} \ || \ \backslash\{y.\text{print}\} \}$,
 asn = $\lambda(x : \text{String}) \rightarrow \lambda(y : \text{IPrint}) \rightarrow \{ \text{print} = \backslash\{x\} = \backslash\{y.\text{print}\} \}$
};

3 Datatype

3.1 Template

BEFORE: **data** $N_D[\overline{T_D}]$ **from** $N_{AI}[\overline{T_{AI}}].S$;

AFTER: **type** $N_D[\overline{T_D}] = \{ \text{accept} : \text{forall } (\overline{T_{AI}} \setminus \overline{T_D}). N_{AI}[\overline{T_{AI}}] \rightarrow S \}$;

NB. Usually $\overline{T_{AI}} \setminus \overline{T_D} = S$.

3.2 Example: List[A]

BEFORE: **sig** ListAlg[A, L] **where**
 nil : L,
 cons : A -> L -> L;
data List[A] **from** ListAlg[A, L].L;

AFTER: **type** ListAlg[A, L] = {
 nil : L,
 cons : A -> L -> L
 };
type List[A] = { **accept** : forall L. ListAlg[A, L] -> L };

4 Creating a Structure

4.1 Template

4.2 Example:

5 Instantiation

5.1 Template

5.2 Example: