Formal Specification F28FS2, Lecture 11
ML as an implementation of Z

Jamie Gabbay

March 11, 2014

Translating the types

Recall the F28PL course on ML. We will now apply this to Z.

Model integers $\mathbb{Z}$, natural numbers $\mathbb{N}$, and nonzero natural numbers $\mathbb{N}_1$ as `int`.

Model powerset $\mathbb{P}T$ as T `list` (only good for finite sets, but often that is enough). So for instance, the Z type $\mathbb{P}\mathbb{P}\mathbb{N}_1$ is modelled in ML as `int list list`.

Lists are ordered and may contain repetitions. This will be fine so long as we only write programs on lists that are not sensitive to order or repititions, they 'might as well' be sets.

Translating the types

Model sequence *seq* T, *iseq* T, and seq_1 T as a list T `list`. So for instance, the Z type $\mathbb{P}(seq(iseq(\mathbb{N})))$ is modelled in ML as `int list list list`.

Model function types as ML function types. So for instance, $\mathbb{N}_1 \rightarrow \mathbb{Z}$ is modelled in ML as `int -> int`.

Model predicates as ML function types to `bool`. So for instance, a binary predicate on numbers such as $<$ is modelled in ML as a term of type `int*int -> int`.

Fibonacci in Z and in ML

Fibonacci numbers specified in Z:

Fibonacci

$$\mathrm{fib} : \mathit{seq}\,\mathbb{N}$$

$$\mathrm{fib}(1) = 1$$
$$\mathrm{fib}(2) = 2$$
$$\forall\, n{:}\mathbb{N} \mid n \geq 2 \bullet \mathrm{fib}(n) = \mathrm{fib}(n-1) + \mathrm{fib}(n-2)$$

Translation to ML:

```
fun fib 1 = 1
  | fib 2 = 2
  | fib n = fib(n-1) + fib(n-2);
```

Ackermann function

Ackermann function specified in Z:

Ackermann

$$\mathrm{ack} : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$$

$$\forall\, n{:}\mathbb{N} \bullet \mathrm{ack}(0, n) = n + 1$$
$$\forall\, m{:}\mathbb{N} \mid m{>}0 \bullet \mathrm{ack}(m, 0) = \mathrm{ack}(m{-}1, 1)$$
$$\forall\, m, n{:}\mathbb{N} \mid m{>}0 \wedge n{>}0 \bullet \mathrm{ack}(m, n) = \mathrm{ack}(m{-}1, \mathrm{ack}(m, n{-}1))$$

Translation to ML:

```
fun ack(0,n) = n+1
  | ack(m,0) = ack(m-1,1)
  | ack(m,n) = ack(m-1,ack(m,n-1));
```

Ackermann function: Z and ML

Compare and contrast the Z spec with the ML function:

The Z spec has quantifiers; the ML function does not. In ML, (universal) quantifiers are implicit in the pattern-matching.

The Z spec has guards, such as $m > 0$ and $n > 0$; the ML function does not. Guards are implicit in the evaluation order in ML.

The Z spec does not have an underlying abstract machine or evaluation order. ML does.

The Z spec is agnostic about the underlying implementation; it does not care if we implement in ML, C, or Brainf*ck—or if we implement at all, or if any implementation even exists. A schema is not a program!

The ML code is still abstract and high-level, but it assumes an underlying machine (and more specifically: top-down left-right eager λ-calculus). For this we sacrifice abstractness.

Sets membership

If $x : T$ and $X : \mathbb{P}T$ then '$x \in X$' is a predicate asserting that x is an element of X. So for instance,
$\text{Andrex} \in \text{FamousBrandNames} : \mathbb{P}\text{BRANDNAME}$ is valid.

Sets membership in ML:

```
fun member x [] = false
  | member x (hd::tl) = (x=hd) orelse member x tl;

val member = fn :  ''a -> ''a list -> bool
member 5 [1,2,3];
val it = false :  bool
member 5 [1,5,5,6];
val it = true :  bool
```

Sets subtraction

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{SetMinusT}} \\
\hline
\setminus : \mathbb{P}\,T \times \mathbb{P}\,T \to \mathbb{P}\,T \\
\hline
\forall X, Y : \mathbb{P}\,T \bullet X \setminus Y = \{x : T \mid x \in X \land x \notin Y\} \\
\hline
\end{array}$$

Translation to ML:

```
fun smin [] Y = []
  | smin (hd::tl) Y = if (member hd Y)
                      then (smin tl Y)
                      else hd::(smin tl Y);

val smin = fn :  ''a list -> ''a list -> ''a list
```

Sets intersection

Just the predicate:

$$X \cap Y = \{x : T \mid x \in X \wedge x \in Y\}$$

Translation to ML:

```
fun sint [] Y = []
  | sint (hd::tl) Y = if (member hd Y)
                      then hd::(sint tl Y)
                      else (sint tl Y);

val sint = fn :  ''a list -> ''a list -> ''a list
```

Sets union

Just the predicate:

$$X \cup Y = \{x : T \mid x \in X \vee x \in Y\}$$

Translation to ML:

```
fun suni [] Y = Y
  | suni (hd::tl) Y = if (member hd Y)
                      then (suni tl Y)
                      else hd::(suni tl Y);
```

```
val suni = fn :  ''a list -> ''a list -> ''a list
```

Or:

```
fun suni' X Y = X @ Y
```

(See also concatenation below.) Compare and contrast the two implementations above.

`suni` and `suni'`

`suni` is relatively slow ($O(n)$ where $n = \#X$), whereas `suni'` is relatively quick (depending on implementation; constant time?).

`suni` tends to eliminate repetitions, e.g. `suni X X` will return `X`. `suni'` tends to create repetition, e.g. `suni' X X` will return two copies of `X` strung together.

So `suni` is good if we care to operate on the result many times, so want an economical representation of the set (no repetitions). `suni'` is good if we do not care about efficiency.

`suni` and `suni'`

Note that 'equality' on `int list` depends on where the `int` came from; if it came from *seq* $\mathbb{N}$ then we care about repetition and ordering, whereas if it came from $\mathbb{P}\mathbb{N}$ then we do not, and two ML lists are 'equal' if they are equal up to repetitions and reordering.

In mathematical computer science, equality is typically a more subtle issue than in pure mathematics.

There may not even be a well-defined notion of equality; e.g. one way to phrase Gödel's incompleteness theorem is that even on the type `unit -> unit`, there is no computable equality.

Stacks and push

Model a stack l of elements of T as $seq\ T$.

A schema to push l:

$$
\begin{array}{|l}
\hline
\textit{push} \\
l, l' : seq\ T \\
hd? : T \\
\hline
l' = \{1 \mapsto hd?\} \cup \{i \mapsto x : l \bullet i + 1 \mapsto x\} \\
\hline
\end{array}
$$

Implementation in ML:

```
fun push hd l = hd::l;
```

Pop

A schema to pop from l:

$$
\begin{array}{|l}
\hline
\textit{pop} \\
l, l' : \textit{seq}\ T \\
hd! : T \\
\hline
\#l > 0 \\
hd! = l(1) \\
l' = \{ i : \mathrm{dom}(l) \mid i > 1 \bullet i{-}1 \mapsto l(i) \} \\
\hline
\end{array}
$$

Implementation in ML:

```
fun pop (hd::tl) = (hd,tl);
```

Concatenation

Recall concatenation:

T cat

$$\frown : seq\ T \times seq T \rightarrow seq\ T$$

$$\forall s, t : seq\ T \bullet$$
$$s \frown t = s \cup \{n \in dom(t) \bullet (n + \#s) \mapsto t(n)\}$$

Implementation in ML (not what I'm looking for):

```
fun concat l1 l2 = l1@l2;
```

Implementation in ML (what I'm looking for):

```
fun conc [] l = l
  | conc (hd::tl) l = hd::(conc tl l);
```

Filtering

Model a predicate on T as a function $T \rightarrow Bool$.

Recall if $L : seq\ T$ and $T' \subseteq T$ then $L \upharpoonright T'$ is the sequence of elements in L that are also in T'.

For example $[tom, dick, harry] \upharpoonright \{tom, harry, jones\} = [tom, harry]$.

Implementation of filtering in ML:

```
fun filter [] P = []
  | filter (hd::tl) P = if (P hd)
                        then hd::(filter tl P)
                        else (filter tl P);

val filter = fn :  'a list -> ('a -> bool) -> 'a list
filter [1,2,3,4] (fn x => not(x=3));
val it = [1,2,4] :  int list
```

Filtering

The set $T' \subseteq T$ became a predicate `P : 'a -> bool`.

Sets T and predicates P are equivalent in Z. Isomorphism given by:

$$\begin{array}{lcl} P & \longmapsto & \{x : T \mid P(x)\} \\ \lambda x : T . x \in T' & \longleftarrow & T' \end{array}$$

ML has two implementations of a predicate on $\mathbb{N}$: as a function `int -> bool`, and as a set `int list`.

Compare and contrast these two: `int list` is an equality type; `int -> bool` is not. `int list` only permits finite sets (such as `[1,2,3]`); `int -> bool` permits, and indeed invites, infinite functions (such as 'is even').

`int -> bool` is the natural model of predicates on $\mathbb{N}$ in Z.

`int list` is the natural model of powerset $\mathbb{N}$ in Z.

Even though in Z, predicates and subsets are isomorphic!

p.s. for the keen: see streams; infinite lists.

Sets by range

```
range m n = if (m>n) then [] else m::(range (m+1) n);

val range = fn :  int -> int -> int list
range 0 5; val it = [0,1,2,3,4,5] :  int list
```

This models the set $0..5 : \mathbb{P}\mathbb{Z}$ (and also $0..5 : \mathbb{P}\mathbb{N}$).

Quantification

Consider

```
fun all [] P = true
  | all (hd::tl) P = (P hd) andalso (all tl P);
val all = fn :  'a list -> ('a -> bool) -> bool
fun exists [] P = false
  | exists (hd::tl) P = (P hd) orelse (exists tl P);
val exists = fn :  'a list -> ('a -> bool) -> bool
```

Q. Translate the predicate $\forall x : 1..10 \bullet x^2 \geq x$ into ML.

A. `all (range 1 10) (fn x => x*x>=x)`.

Divisibility

$x|y$ (x divides y) when $\exists z : \mathbb{N} \mid z \leq y \bullet z * x = y$.

In ML:

```
fun divides x y = exists (range 0 y) (fn z => z*x=y)
```

```
- divides 4 10;
val it = false :  bool
- divides 5 10;
val it = true :  bool
```

Prime

y is prime when $\forall x : \mathbb{N} \mid x | y \bullet x = 1 \vee x = y$.

In ML:

```
fun prime y = all (range 2 (y-1)) (fn x => not
(divides x y));
```

```
- prime 1;
val it = true :  bool
- prime 2;
val it = true :  bool
- prime 3;
val it = true :  bool
- prime 4;
val it = false :  bool
```

Arguably slight bug in this; 1 is not generally considered a prime number.

Map

Recall `map : ('a -> 'b) -> 'a list -> 'b list`.

In ML:

```
fun map f [] = []
  | map f (hd::tl) = (f hd)::(map f tl);
```

Exercise: specify what ML does as a Z schema, thus

$$\begin{array}{|l}
\hline
\textit{map} \\
\hline
map : (T \to T') \to seq\ T \to seq\ T' \\
\hline
\ldots \\
\hline
\end{array}$$

`map` is the primitive of supercomputer architecture (highly parallel, stream processor based); guarantee of non-interference given by the ML language itself, which is purely functional (kind of).

Exercises

Express the following in ML:

1. The elements of $\mathcal{X} : \mathbb{PPZ}$ are pairwise disjoint (that is, $\forall X, Y : \mathcal{X} \bullet X = Y \vee X \cap Y = \varnothing$).
2. $\mathcal{X}$ covers X (that is, $\bigcup \mathcal{X} = X$).
3. $\mathcal{X} : \mathbb{PPZ}$ is a partition of $X : \mathbb{PZ}$ (that is, $\mathcal{X}$ covers X and its elements are pairwise disjoint).
4. Using `filter` and `divides` or otherwise, write a function which inputs x and returns the list of prime numbers from 1 to x (see the Sieve of Eratosthenes).

Exercises

Express the following in ML:

1. An ML type to model $\mathbb{N} \leftrightarrow \mathbb{N} = \mathbb{P}(\mathbb{N} \times \mathbb{N})$.
2. A function to check that `x` is in the model of this type and not, say, of $\mathbb{Z} \leftrightarrow \mathbb{Z}$.
3. Domain restriction $S \lhd f$ where $S : \mathbb{P}\mathbb{N}$ (modelled as a set) and $f : \mathbb{N} \leftrightarrow \mathbb{N}$.
4. Domain restriction $S \lhd f$ where $S : \mathbb{P}\mathbb{N}$ (modelled as a predicate) and $f : \mathbb{N} \leftrightarrow \mathbb{N}$.
5. Range anti-restriction $f \ntriangleright S$.