# Formal Specification F28FS2, Lecture 13
# Totalising schema in ML

Jamie Gabbay

March 12, 2014

# Recall: Pop

Recall this schema to pop from $l$:

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

Recall its implementation in ML:

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

# Pop

The specification is partial.

The function satisfies the specification. However, the specification does not say what should happen when $l$ is empty.

We could return a default value. However, $T$ is an abstract type; which default value to put in $hd$!?

Better to return an error.

Declare a type $MESSAGE ::= success \mid popEmptyError$.

# Pop

*pop*

$l, l' : seq\ T$

$hd! : T$

$message! : MESSAGE$

---

$\#l > 0 \land l' = \{i : \mathrm{dom}(l) \mid i > 1 \bullet i - 1 \mapsto l(i)\} \land hd! = l(1)$

$message! = success$

*popEmpty*

$l, l' : seq\ T$

$message! : MESSAGE$

---

$\#l = 0 \land l' = l$

$message! = popEmptyError$

$totalPop \mathrel{\widehat=} pop \land popEmpty$

# Modelling *pop* in ML

We could model *totalPop* literally; it returns *hd*! and *message*! and $l'$, thus returns a 3-tuple.

```
datatype MESSAGE = success | popEmptyError;
```

```
fun pop (hd::tl) = (hd,tl,success)
  | pop []       = (0,[],popEmptyError);
val pop = fn :  int list -> int * int list * MESSAGE
```

I don't like this: the 3-tuples are unattractive; but worse, we have lost polymorphism because ML insists we return something in *hd*! in the *popEmpty* case. I chose 0, thus effectively forcing us to choose $T{=}\texttt{int}$.

Note that *totalPop* does not specify *hd*! in the empty case (see *popEmpty*; *hd*! is not even in the schema variables). But ML cannot do that; something has to go into *hd*! . . . or does it?

# Modelling *pop* in ML, version 2.0: exceptions

Different model. Declare an ML exception.

```
exception popExn

fun pop (hd::tl) = (hd,tl)
  | pop []       = raise popExn;
val pop = fn :  'a list -> 'a * 'a list
```

Exceptions can be handled. To recreate our previous implementation:

```
fun pop' l = ((fn (x,y) => (x,y,success)) (pop l))
             handle popExn => (0,l,popEmptyError);
val pop' = fn :  int list -> int * int list * MESSAGE
```