On Sat, Nov 28, 2020 at 3:16 AM Michael Kay <mike@saxonica.com> wrote:

> Tuple types - or Record types as I'm now proposing to call them - possibly provide a way forward on this. For example the math functions could be declared as an instance of a record type
>
> record (pi as function() as xs:double,
> sin as function(xs:double) as xs:double,
> ... etc)

Thank you Dr. Kay for your positive response.

> and if $math is statically known to conform to this type, then $math?sin is statically known
> to be a function with a particular signature. (But note, it's $math?sin, not $math?sin#1).

It is amusing how easy this is to do:

Let's have a map `$Strings` and one of its members is the map `$replace`:

```
let $replace := map {
            '_3' : replace#3,
            '_4' : replace#4
            }
return
   $replace?_3 ( 'abbbcdbbbba', 'b', 'B')
```

produces exactly the wanted result:

```
aBBBcdBBBBa
```

We certainly can add a little bit of language support, for example to allow the second argument of ? to accept the # character. Then we could have even this:

```
let $replace := map {
            '#3' : replace#3,
            '#4' : replace#4
            }
return
   $replace?#3 ( 'abbbcdbbbba', 'b', 'B')
```

We can go even a little bit further and specify that the way to evaluate this expression:

```
  $replace('abbbcdbbbba', 'b', 'B')
```

as:

"***When a map contains just several overloads of a function (such as '#3', '#4') and is called as a function without specifying arity, then the arity of the actual call is used to determine the member of the map that will be the actual function to be invoked***."

In the above case the arity used in the function invocation is 3, thus `replace#3` will be invoked and we get the same correct result:

```
aBBBcdBBBBa
```

> it's a bit hard to see how to introduce it alongside existing mechanism without ending up with
> a rather clumsy mixture of different approaches coexisting

Grouping functions within maps is very flexible. A function can be referenced by a map, and in the same time it can be referenced directly (globally) as in all versions of XPath up to `3.1`.
Thus there is no backwards compatibility problem.

Versions of XPath > `3.1` will provide the `$fn:Functions` map which represents a hierarchy (actually a forest) that is a grouping of all functions provided in the F & O standard. Starting with the next version of XPath, people will use this "`SysMap`" as it is easy and intuitive to learn and navigate (a semantic hierarchy) and minimizes the possibility of name and signature conflicts.

So there is not going to be a "clumsy mixture" because there is not going to be a mixture at all -- code written in the older versions uses the global referencing, code written with the new versions relies heavily on the convenience of the `SysMap` functions grouping.

I would like to thank all the readers for their support and would appreciate your further comments and suggestions.

Thanks,
Dimitre