What is the measure?

Augusto Ribeiro
ari@iha.dk

November 8, 2008

1 Termination Proof Obligations

Termination proof obligations are like ordinary proof obligations. If they are proved valid then it means that the recursive function that it refers too will terminate. They normally impose that the recursive argument is decreasing in each call.

For the *factorial* function:

```
fact : nat -> nat
fact(n) ==
  if n = 0 then 1
  else n*fact(n-1)
```

The proof obligation would be:

$$\forall n \in \mathcal{N} : n > n - 1$$

Which is trivially valid and that means that factorial finishes.

2 Measure

This kind of reasoning is only valid if we are operating over the $>$ relation in natural numbers. It can be seen that if a argument is always decreasing in the $>$ of integers, it does not mean the function finishes because the $>$ relation for integers has no lower limiter (like 0 in the case of $>$ relation in natural numbers). The $>$ relation of natural numbers is called a well founded relation and in termination proof obligations these are the only ones that can be used.

In VDMTools, the $>$ relation from natural number is used for every termination proof obligation. So in case of a function which has a domain different from nat a mapping from the domain data type to naturals is needed. This mapping function is called the **measure**.

The following example has a sequence as domain:

```
reverse : seq of char -> seq of char
reverse(str) ==
  if str = []
  then []
  else reverse(tl str)^[hd str];
```

A possible measure for this function could be the length of the list.

```
reverse_m : seq of char -> nat
reverse_m(str) == len str
```

The choice of the measure for a recursive functions is indicated with the keyword `measure`.

```
reverse : seq of char -> seq of char
reverse(str) ==
 ...
measure reverse_m;
```

This choice would give origin to the following proof obligations.

$$\forall str \in \text{seq of char} : \text{reverse\_m}(str) > \text{reverse\_m}(tl\ str)$$

3 Lexicographic Orders

If you have time, you may consider implementing the following:

Some recursive functions require a different approach using a lexicographic order instead of the simple $>$ relation. A lexicographic order is a product of two order relations and it is defined as:

$$(a, b) \leq (a', b') \text{ if and only if } a < a' \text{ or } (a = a' \text{ and } b \leq b')$$

The next example illustrates a function which is the breath-first traversal of graph that needs a lexigraphic order to be used in order to prove its termination. In this case the result of the measure has to be a tuple of `nat` instead.

```
types
Graph = map nat to seq of nat
inv g == forall i in set dom g & elems g(i) subset dom g;

functions
depthf : seq of nat * Graph * seq of nat -> seq of nat
depthf(l,g,vis) ==
  cases l:
  [] -> reverse vis,
  h^[t] -> if h in set vis
               then depthf(t,g,vis)
               else depthf(g(h)^t,g,h^vis)
pre elems l subset dom g;
```

A measure for this function would be:

```
meas_depthf : seq of nat * Graph * seq of nat -> nat * nat
meas_depthf(l,g,vis) ==
  mk_(card dom graph - len vis, len l)
```

The use of a lexicographic function in the proof obligations is marked `LEX`. Using the above measure, the proof obligation that is generated is:

```
forall l:seq of nat, g:Graph, vis:seq of nat &
         not(l = []) and h in set vis and t = tl l =>
           meas_depthf(l,g,vis) (LEX2 >)  meas_dephf(t,g,vis)
         and
         not(l=[]) and not(h in set vis t) and t = tl l and h = hd l =>
           meas_depthf(l,g,vis) (LEX2 >) meas_depthf(g(h)^t,g,h^vis)
```

The number next to LEX indicates the arity of the tuple to compare.