Termination combinators forever

Simon Peyton Jones (Microsoft Research)
Max Bolingbroke (University of Cambridge)

2011

Termination testing is useful

- ...in compilers
- ...in supercompilers
- ...in theorem provers

It's a useful black box.

- But it should be modularly separated from the rest of your compiler/theorem prover/whatever
- (The typical reality is otherwise.)

The problem

- Online termination detection
- Given a sequence of values, x0, x1, x2...
- ...presented one by one...
- ...yell "stop" if it looks as if the sequence might be diverging
- Guarantee never to let through an infinite sequence
- Delay "stop" as long as possible
- "Values" includes pairs, strings, trees....

Concretely

```
data TTest a
testList :: TTest a -> [a] -> Bool
```

- Postpone: where do TTests come from?
- Note: testList is inherently inefficient for the "present one at a time" situation

Better...

```
data History a
initHistory :: TTest a -> History a

data TestResult a = Stop | Continue (History a)
test :: History a -> a -> TestResult a
```

- Intuitively the History accumulates (some abstraction of) the values seem so far

Creating TTests

- The goal: a library that makes it easy to construct values of type TTest a, that
  - Are definitely sound: they do not admit infinite sequences
  - Are lenient as possible: they do not blow the whistle too soon

Creating TTests

```
intT      :: TTest Int
boolT     :: TTest Bool
pairT     :: TTest a  -> TTest b -> TTest (a, b)
eitherT   :: TTest a  -> TTest b -> TTest (Either a b)
wrapT     :: (a -> b) -> TTest b -> TTest a
```

- Just the usual type-directed combinator library

Implementing TTests

- How do we implement a TTest?
- Find a strictly-decreasing measure bounded below.
- This is VERY INCONVENIENT in many cases. Think about a sequence of syntax trees.
- Well-studied problem, standard approach: use a well-quasi order (WQO).

Well-quasi orders

Definition

A transitive binary relation $\leq$ is a WQO

iff

For any infinite sequence

$x_0, x_1, x_2....$

there exists $i<j$ st $x_i \leq x_j$

Theorem: every WQO is reflexive

From WQOs to TTests

```
newtype TTest a = TT (a -> a -> Bool)
data History a = H (a->a->Bool) [a]

initHistory :: TTest a -> History a
initHistory (TT wqo) = H wqo []

test :: History a -> a -> TestResult a
test (H wqo vs) v
  | any (`wqo` v) vs = Stop
  | otherwise        = Continue (H wqo (v:vs))
```

- New goal: a (trusted) library that helps you to define (sparse) WQOs, that really are WQOs

Finite sets

```
finiteT :: Finite a => TTest a
finiteT = TT (==)

class Eq a => Finite a where
   elements :: [a]
```

- Is (==) a WQO on finite sets? Yes.
- Odd; we don't use the methods of Finite
- Instead, Finite is really a **proof obligation**:
  - There are only a finite number of elements of a
  - (==) is reflexive

Sums

```
eitherT:: TTest a -> TTest b -> TTest (Either a b)
eitherT (TT wqo_a) (TT wqo_b) = TT wqo
   where
     (Left x)  `wqo` (Left y)  = x `wqo_a` y
     (Right x) `wqo` (Right y) = x `wqo_b` y
     _ `wqo` _ = False
```

- Is this a WQO? Why?

Products

```
pairT:: TTest a -> TTest b -> TTest (a,b)
pairT (TT wqo_a) (TT wqo_b) = TT wqo
  where
   (x1,x2) `wqo` (y1,y2) = ....
```

Products

```
pairT:: TTest a -> TTest b -> TTest (a,b)
pairT (TT wqo_a) (TT wqo_b) = TT wqo
  where
   (x1,x2) `wqo` (y1,y2) = x1 `wqo_a` y1
                          && x2 `wqo_b` y2
```

- But is this a WQO?
- For any infinite sequence (x0,y0), (x1,y1), ... can we be sure there is an i<j, st
  $xi \le xj$, and $yi \le yj$
  ?
- Yes, and the proof is both simple and beautiful

Back to WQOs

Theorem. If ($\leq$) is a WQO, then
for any infinite sequence $x_0, x_1, x_2, \ldots$
there is a finite N such that
for any i>N
there is a j>i
such that $x_i \leq x_j$

**That is, after some point N,
every x_i is $\leq$ a later x_j**

Proof: Consider $\{x_i \mid \nexists j>i.\ x_i \leq x_j\}$

Corollary: every infinite sequence has a chain
$x_{i1} \leq x_{i2} \leq x_{i3} \leq \ldots$

Cofunctors

```
wrapT:: (b->a) -> TTest a -> TTest b
wrapT f (TT wqo_a) = TT wqo_b
  where
    x `wqo_b` y = f x `wqo_a` f y

instance CoFunctor TTest where
  cofmap = wrapT
```

- Exercise: modify the implementation of TTest and History to avoid the repeated re-application of f.

Even more fun...recursive types

```
unwrap :: [a] -> Either () (a, [a])
unwrap []       = Left ()
unwrap (x:xs) = Right (x,xs)

listT :: forall a. TTest a -> TTest [a]
listT telt = tlist
  where
    tlist :: TTest [a]
    tlist = cofmap unwrap $
              eitherT finiteT
                      (pairT telt tlist)
```

- The types are right
- We are only using library combinators
- Does it work?

Lists

```
unwrap :: [a] -> Either () (a, [a])
unwrap []      = Left ()
unwrap (x:xs) = Right (x,xs)

listT :: forall a. TTest a -> TTest [a]
listT telt = tlist
  where
    tlist :: TTest [a]
    tlist = cofmap unwrap $
            eitherT finiteT
                    (pairT telt tlist)
```

- Consider [], [1], [1,1], [1,1,1], [1,1,1,1], ...
- An infinite sequence... accepted!
- What has gone wrong?

The problem

- We assumed that tlist was WQO when proving that it is a WQO!

```
tlist :: TTest [a]
tlist = cofmap unwrap $
        eitherT finiteT
                (pairT telt tlist)
```

- Sort-of solution: make the combinators strict, so tlist is bottom
- ...But we still want a termination checker for lists!

Homeomorphic embedding

```
wqoL :: WQO a -> [a] -> [a] -> Bool
wqoL we [] ys      = True
wqoL we (x:xs) [] = False
wqoL we (x:xs) (y:ys)
  =  (x `we` y && wqoL we xs ys)
  || wqoL we (x:xs) ys
```

"**Couple**": See if they match at the root

"**Dive**": See if the first arg matches inside the recursive component of the second arg

- This actually is a WQO
- The proof is not obvious, at all
- Q1: find an elegant proof

Lists

```
recT :: (t -> [t])
     -> TTest t
     -> TTest t
```

Function to get the "recursive children" of a t-value

A "couple" tester: match at the root

```
listT :: TTest t -> TTest [t]
listT telt = tlist
  where
    tlist :: TTest [a]
    tlist = recT kids $
              cofmap unwrap $
              eitherT finiteT
                      (pairT telt tlist)
    kids [] = []
    kids (x:xs) = [xs]
```

Lists

```
recT :: (t -> [t])
     -> TTest t
     -> TTest t
recT kids ~(TT wqo_top)= TT wqo
  where
    x `wqo` y = x `wqo_top` y)
              || any (x `wqo`) (kids y)
```

Function to get the "recursive children" of a t-value

A "couple" tester: match at the root

Lists

```
recT :: (t -> [t])
     -> TTest t
     -> TTest t
recT kids ~(TT wqo_top)= TT wqo
  where
    x `wqo` y = x `wqo_top` y)
                || any (x `wqo`) (kids y)
```

Function to get the "recursive children" of a t-value

A "couple" tester: match at the root

- Q2: is this the best formulation?
- Q3: what is the proof obligation for "kids"
- Q4: solve nasty interaction with cofmap
- Q5: Elucidate relationship to R+

Summary

- A combinator library for online termination testing
- A useful black box, never previously abstracted out as such
- Encapsulates tricky theorems inside a nice, compositional interface