CATCH\textsuperscript{1}: Case and Termination Checking for Haskell

Neil Mitchell

(supervised by Colin Runciman)

\textsuperscript{1} Name courtesy of Mike Dodds
Termination Checkers

Q) Does function $f$ terminate?
A) \{Yes, Don’t know\}

- Typically look for decreasing size
  - Primitive recursive
  - Walther recursion
  - Size change termination
Does this terminate?

\[ \text{fib} :: \text{Integer} \rightarrow \text{Integer} \]
\[ \text{fib}(1) = 1 \]
\[ \text{fib}(2) = 1 \]
\[ \text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2) \]
\[ \text{fib}(0) = \bot^{NT} \]
Remember the value!

- A function only stops terminating when it is given a *value*
- Perhaps the question is wrong:
  
  **Q)** Given a function $f$ and a value $x$, does $f(x)$ terminate?
  
  **Q)** Given a function $f$, for what values of $x$ does $f(x)$ terminate?
But that’s wrong…

```
fib n | n <= 0 =
      error "bad programmer!"
```

- A function should *never* non-terminate
- It *should* give an helpful error message
- There may be a few exceptions
  - But probably things that can’t be proved
  - i.e. A Turing machine simulator
Haskell is:
- A functional programming language
- Lazy – not strict

Only evaluates what is required

Lazy allows:
- Infinite data structures
Productivity

\[ [1..] = [1, 2, 3, 4, 5, 6, \ldots] \]

- Not terminating
- But is *productive*
  - Always another element
  - Time to generate “next result” is always finite
The blame game

- last [1..] is ⊥^NT
- last is a useful function
- [1..] is a useful value

Who is at fault?
- The caller of last
A Lazy Termination Checker

- All data/functions must be productive
- Can easily encode termination

```haskell
isTerm :: [a] -> Bool
isTerm []     = True
isTerm (x:xs) = isTerm xs
```
NF, WHNF

- Normal Form (NF)
  - Fully defined data structure
  - Possibly infinite
  - `value{*}`

- Weak Head Normal Form (WHNF)
  - Outer lump is a constructor
  - `value{?}`

- `value{*} \Rightarrow value{?}`
last \ x = \text{case } x \text{ of}

\ (:) \to \text{case } x.\ tl \text{ of}

\ [ ] \to x.\ hd

\ (:) \to \text{last } x.\ tl

(last x)\{?\} = x\{[ ]\} \vee (x.\ tl\{:\} \vee (x.\ hd\{?\}))

\ (last x)\{?\} = x\{[ ]\} \vee x.\ tl\{[ ]\} \vee (last x.\ tl)\{?\}

= x\{[ ]\} \vee x.\ tl\{[ ]\} \vee x.\ tl.\ tl\{[ ]\} \vee ...

= \exists i \in L(tl^*), x.\ i\{[ ]\}

= x.\ tl^3\{[ ]\}

(last x)\{^*\} = (last x)\{?\} \wedge (x\{[ ]\} \vee x.\ tl\{[ ]\} \vee (last x.\ tl)\{^*\})

= x.\ tl^3\{[ ]\}
And the result:

\[(\text{last } x)\{\ast\} = x\{\ast\} \land x.tl \equiv \{[]\}\]

- x is defined
- x has a [], x is finite

A nice result ☺
Ackermann’s Function

data a Nat = S Nat | Z
ack Z n = S n
ack (S m) Z = ack m (S Z)
ack (S m) (S n) = ack m (ack (S m) n)

- (ack m n) {?} = m.p³{Z} ^ m{*} ^ n{*}
- ack 1 ∞ = ? (answer is ∞)
- ack ∞ 1 = ⊥NT
Conclusion

- What lazy termination might mean
  - Productivity
  - Constraints on arguments
  - WHNF vs NF

- Lots to do!
  - Check it
  - Prove it
  - Implement it